



## Towards We-Government: Collective and participative approaches for addressing local policy challenges

Grant Agreement number: 693514

### Deliverable

#### D3.1

## Consolidated System Architecture

Project co-funded by the European Commission within H2020-EURO-2014-2015/H2020-EURO-6-2015		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

## Author List

Organisation	Name	Contact Information
UniTo	Alessio Antonini	<a href="mailto:antonini@di.unito.it">antonini@di.unito.it</a>
UniTo	Liliana Ardissono	<a href="mailto:liliana.ardissono@unito.it">liliana.ardissono@unito.it</a>
UniTo	Matteo Baldoni	<a href="mailto:baldoni@di.unito.it">baldoni@di.unito.it</a>
UniTo	Cristina Baroglio	<a href="mailto:baroglio@di.unito.it">baroglio@di.unito.it</a>
LiquidFeedback	Jan Behrens	<a href="mailto:jan.behrens@flexiguided.de">jan.behrens@flexiguided.de</a>
UniTo	Guido Boella	<a href="mailto:boella@di.unito.it">boella@di.unito.it</a>
Funka	Andreas Cederbom	<a href="mailto:andreas.cederbom@funka.com">andreas.cederbom@funka.com</a>
UniTo	Egidio Dansero	<a href="mailto:egidio.dansero@unito.it">egidio.dansero@unito.it</a>
UniTo	Luigi Di Caro	<a href="mailto:dicaro@di.unito.it">dicaro@di.unito.it</a>
INFALIA	Sotiris Diplaris	<a href="mailto:sdip@infalia.com">sdip@infalia.com</a>
INFALIA	Elena Dougia	<a href="mailto:elena@infalia.com">elena@infalia.com</a>
UCL	Claire Ellul	<a href="mailto:c.ellul@ucl.ac.uk">c.ellul@ucl.ac.uk</a>
Empirica	Karsten Gareis	<a href="mailto:karsten.gaeris@empirica.com">karsten.gaeris@empirica.com</a>
UCL	Carles Boïls Gisbert	<a href="mailto:c.gisbert@ucl.ac.uk">c.gisbert@ucl.ac.uk</a>
Funka	Tommy Feldt	<a href="mailto:tommy.feldt@funka.com">tommy.feldt@funka.com</a>
MfC	Louise Francis	<a href="mailto:l.francis@mappingforchange.org.uk">l.francis@mappingforchange.org.uk</a>
LiquidFeedback	Axel Kistner	<a href="mailto:axel.kistner@flexiguided.de">axel.kistner@flexiguided.de</a>
Empirica	Lutz Kubitschke	<a href="mailto:lutz.kubitschke@empirica.com">lutz.kubitschke@empirica.com</a>
UniTo	Maurizio Lucenteforte	<a href="mailto:lucenteforte@di.unito.it">lucenteforte@di.unito.it</a>
UniTo	Lucia Lupi	<a href="mailto:lupi@di.unito.it">lupi@di.unito.it</a>
UniTo	Roberto Micalizio	<a href="mailto:micalizio@di.unito.it">micalizio@di.unito.it</a>
INFALIA	Alexandros Mokkas	<a href="mailto:mokkas@infalia.com">mokkas@infalia.com</a>
INFALIA	Spiros Nikolopoulos	<a href="mailto:snik@infalia.com">snik@infalia.com</a>
LiquidFeedback	Andreas Nitsche	<a href="mailto:andreas.nitsche@flexiguided.de">andreas.nitsche@flexiguided.de</a>
UHEI	Alexey Noskov	<a href="mailto:noskov@uni-heidelberg.de">noskov@uni-heidelberg.de</a>
MfC	Julius Osokinas	<a href="mailto:j.osokinas@mappingforchange.org.uk">j.osokinas@mappingforchange.org.uk</a>
INFALIA	Symeon	<a href="mailto:spap@infalia.com">spap@infalia.com</a>

	Papadopoulos	
UCL	Sophie Papadopoulou	<a href="mailto:sofia.papadopoulou.15@ucl.ac.uk">sofia.papadopoulou.15@ucl.ac.uk</a>
UniTo	Giovanna Petrone	<a href="mailto:petrone@di.unito.it">petrone@di.unito.it</a>
UHEI	Adam Rousell	<a href="mailto:adam.rousell@uni-heidelberg.de">adam.rousell@uni-heidelberg.de</a>
UniTo	Adriano Savoca	<a href="mailto:savoca@di.unito.it">savoca@di.unito.it</a>
UniTo	Claudio Schifanella	<a href="mailto:schi@di.unito.it">schi@di.unito.it</a>
UniTo	Marino Segnan	<a href="mailto:segnan@di.unito.it">segnan@di.unito.it</a>
UCL	Artemis Skarlatidou	<a href="mailto:a.skarlatidou@ucl.ac.uk">a.skarlatidou@ucl.ac.uk</a>
LiquidFeedback	Björn Swierczek	<a href="mailto:b.swierczek@flexiguided.de">b.swierczek@flexiguided.de</a>
INFALIA	Ioannis Tsampoulatidis	<a href="mailto:itsam@infalia.com">itsam@infalia.com</a>
INFALIA	Stefanos Vrochidis	<a href="mailto:svro@infalia.com">svro@infalia.com</a>

## Status, Abstract, Keywords, Statement of originality

<b>Deliverable Title</b>	Consolidated System Architecture
<b>Deliverable No.</b>	3.1
<b>Dissemination level:</b>	Public
<b>Leading partner</b>	UniTo
<b>Participating partners</b>	LiquidFeedback, UHei, UCL, Mapping for Change, Funka, INFALIA
<b>Contractual date of delivery:</b>	31 January 2017
<b>Actual date of delivery:</b>	15 February 2017
<b>Work Package:</b>	WP3 Agile Development of the WeGovNow platform
<b>Type:</b>	Report
<b>Approval Status:</b>	Final
<b>Version:</b>	1.0
<b>Abstract</b> <p>This document details the consolidated architecture of the WeGovNow platform prototype. In doing so, it builds upon the conceptual and methodological framework for the WeGovNow project presented in a dedicated deliverable (D1.2)</p>	
<b>Keywords</b> <p>Architecture, web applications, user management, software integration, API, OAuth 2.0, user authentication, system logging</p>	

### Statement of originality

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

## History

Version	Date	Reason	Revised by
	1/12/2016	Creation of the initial document structure	Alessio Antonini, Claudio Schifanella
	12/12/2016	2.1 3.5 4.7 4.9 4.11 6.3 7.1	Andreas Nitsche
	14/01/2017	3.3	Claudio Schifanella
	15/01/2017	2	Alessio Antonini
	18/01/2017	Appendix 3	Ioannis Tsampoulatidis
	27/01/2017	Appendix 7.2 and update of quality section in main text	Adam Rousell
	30/01/2017	Requirements and technical scenarios	Alessio Antonini
	31/01/2017	Temporal indexing	Claudio Schifanella
	31/01/2017	Service scenarios and typos	Louise Francis
	31/01/2017	Look and Feel, Design Guide, Connecting requirements, WeGovNow toolbox	Alessio Antonini
	1/02/2017	Introduction	Alessio Antonini
	2/02/2017	Conclusions, Integration diagram	Alessio Antonini
	13/02/2017	ImproveMyCity, Trusted Marketplace, Bilateral communication with other components, Testing introduction and overall feedback	I.Tsampoulatidis, A.Mokkas, S.Vrochidis, E. Dougia, A.Papadopoulos, S.Diplaris, S.Nikolopoulos
	14/02/2017	Internal review	Alessio Antonini, Lutz Kubitschke
	17/02/2017	Finalisation	Alessio Antonini, Karsten Gareis

## Table of Contents

1	Introduction .....	11
2	WeGovNow overall architecture.....	12
2.1	Unified WeGovNow Authentication System.....	13
2.1.1	Secure authentication and authorization.....	15
2.1.2	Dynamic client registration .....	16
2.2	Application Discovery Service .....	16
2.3	Style Service.....	16
2.4	NavigationBar .....	17
2.5	Centralised User Profile .....	17
2.6	Centralised Settings .....	18
2.6.1	Accessibility Settings .....	19
2.7	Centralised Activity Logger .....	20
2.7.1	OnToMap Logger architecture and APIs.....	21
2.8	Landing page.....	25
2.8.1	Structure.....	25
2.8.2	User's area.....	26
2.8.3	AreaView.....	26
2.8.4	Authentication support .....	26
2.9	InputMap .....	26
2.9.1	Use.....	27
2.9.2	Data sources .....	28
2.9.3	TileServer.....	28
2.9.4	Area/TileIndex .....	28
2.10	AreaView .....	29
2.10.1	Structure and use of AreaViewer .....	29
2.10.2	Notifications .....	30
2.10.3	Data Sources.....	30
3	WeGovNow components .....	31
3.1	GeoKey.....	33
3.1.1	Extending GeoKey for WeGovNow.....	34
3.1.2	Adding Social Media Connectivity .....	34
3.1.3	Adding Material Design .....	35
3.2	Community Maps .....	37
3.2.1	Integrating with UWUM .....	37
3.2.2	Adding Material Design .....	37
3.3	FirstLife .....	37
3.3.1	FirstLife backend .....	38
3.3.2	The unified data model .....	39

3.3.3	FirstLife client .....	40
3.3.4	Working with dynamic maps .....	41
3.4	Improve My City .....	43
3.4.1	ImproveMyCity API calls and security .....	46
3.4.2	Extending ImproveMyCity for WeGovNow .....	48
3.5	LiquidFeedback .....	52
3.5.1	LiquidFeedback Core extensions .....	54
3.5.2	Planned REST API including geo-spatial queries .....	55
3.5.3	Further details on implementation .....	56
3.5.4	UWUM and integration framework .....	56
3.6	OnToMap .....	56
3.6.1	OnToMap backend architecture .....	57
3.6.2	Data retrieval and integration .....	58
3.6.3	OnToMap data retrieval API .....	59
3.7	Trusted Marketplace .....	62
4	Component integration .....	64
4.1	Community Maps/GeoKey ⇄ WeGovNow Components via OnToMap and AreaViewer .....	65
4.2	FirstLife ⇄ LiquidFeedback and FirstLife ⇄ ImproveMyCity .....	65
4.3	FirstLife ⇄ OnToMap .....	67
4.4	ImproveMyCity ⇄ LiquidFeedback .....	67
4.5	LiquidFeedback ⇄ OnToMap and ImproveMyCity ⇄ OnToMap .....	68
4.6	OnToMap ⇄ TrustedMarketplace .....	69
5	Technology and user driven scenarios .....	69
5.1	Technology driven usage scenarios .....	70
5.1.1	User setup .....	70
5.1.2	Data entry .....	71
5.2	Service scenarios .....	71
5.3	Connecting requirements and platform development .....	72
5.4	Assessment of use cases and user requirements .....	73
6	Modularity, extendibility and challenges of WeGovNow platform .....	74
6.1	Testing .....	75
6.2	WeGovNow monitoring .....	76
6.3	Connecting with WeGovNow core .....	76
6.4	Connecting with WeGovNow data .....	77
6.4.1	Look and Feel .....	82
7	WeGovNow toolbox .....	83
7.1	Geographical support .....	83
7.2	Quality assessment .....	86
7.3	Data sources .....	87

7.4	Temporal indexing of urban entities .....	91
7.5	Design Guide .....	92
8	Conclusions .....	93
9	Appendices .....	95

## List of Exhibits

Exhibit 1: WeGovNow core features provides the environment to integrate modular components accordingly to the specific needs of each WeGovNow instance .....	13
Exhibit 2: Login page delivered by UWUM.....	14
Exhibit 3: OAuth 2.0 Authorization Code flow .....	15
Exhibit 4: UWUM registration form initialises the user profile with the form data entered during registration .....	18
Exhibit 5: UWUM registration form initialises the user settings .....	19
Exhibit 6: Interaction between a WeGovNow component and the OnToMap Logger .....	21
Exhibit 7: Translation of a user activity event to the format managed by the OnToMap Logger .....	22
Exhibit 8: Structure of the user activity events accepted by the OnToMap Logger .....	24
Exhibit 9: LandingPage screenshot.....	25
Exhibit 10: LandingPage structure.....	26
Exhibit 11: Input Map.....	27
Exhibit 12: InputMap constitutes of a cartographic layer and a vector layer of relevant geographical entities.....	28
Exhibit 13: AreaViewer is composed by three main layers connected to three data sources specifically designed to reduce the computational costs at client side, and cacheable .....	30
Exhibit 14: WeGovNow components are interconnected through API and shared functionalities specifically developed to support a distributed orchestration of features.....	31
Exhibit 15: GeoKey architecture and main components.....	33
Exhibit 16: GeoKey's key features .....	34
Exhibit 17: GeoKey's projects overview with Material Design .....	36
Exhibit 18: GeoKey's project creation with Material Design .....	36
Exhibit 19: FirstLife client/server architecture and main components .....	38
Exhibit 20: Specialised entities in FirstLife and their relations .....	39
Exhibit 21: FirstLife database schema implements a hierarchical structure .....	40
Exhibit 22: Use case for searching and add a new description to an entity .....	40
Exhibit 23: Information flow of the use case in Exhibit 22, in red the expected delays in client components due to asynchronous calls to the resource server.....	41
Exhibit 24: The change of Bounding Box triggers a query toward FirstLife backend API, the response is consolidated with the local memory of firstlife client before being rendered on the map....	42
Exhibit 25: Entity and category filters on FirstLife web client .....	43

Exhibit 26: Tile id is used to index the retrieved data making much more efficient the management at client side .....	43
Exhibit 27: ImproveMyCity overall architecture as part of the Joomla! framework.....	44
Exhibit 28: ImproveMyCity package contents.....	45
Exhibit 29: ImproveMyCity API request data flow .....	47
Exhibit 30: Screenshot of ImproveMyCity with Navigation Bar and Material Design applied .....	49
Exhibit 31: Extension of ImproveMyCity architecture towards WeGovNow integration.....	50
Exhibit 32: New approach on ImproveMyCity API methods invocation for WeGovNow.....	51
Exhibit 33: The structured proposition development and decision making process in LiquidFeedback	52
Exhibit 34: Dynamic division of labour: Delegations for units, areas and issues.....	53
Exhibit 35: Architecture of LiquidFeedback as WeGovNow component .....	54
Exhibit 36: Architecture of the OnToMap backend.....	57
Exhibit 37: Sample result of the invocation of OnToMap Data Retrieval API to get the concepts of the OnToMap Ontology.....	60
Exhibit 38: Sample result of the invocation of OnToMap Data Retrieval API to get the instances of a concept of the OnToMap Ontology.....	61
Exhibit 39: High level architecture of Trusted Marketplace.....	63
Exhibit 40: Bilateral integration between FirstLife and LiquidFeedback, and between FirstLife and ImproveMyCity .....	66
Exhibit 41: WeGovNow use case development approach .....	72
Exhibit 42: Integration of concept “RentedApartment” into the OnToMap Ontology .....	78
Exhibit 43: Rule representing the mapping between the schema of an application App1 and the schema of OnToMap .....	80
Exhibit 44: JSON schema specifying the concept translation rules to OnToMap Logger format .....	80
Exhibit 45: Surface distance between two points .....	84
Exhibit 46: Fractal indexing using the Z-Order Curve .....	84
Exhibit 47: Portion of the OnToMap Ontology, focusing on concept “SchemaThing”, depicted in boldface for easy recognition.....	89
Exhibit 48: Portion of the OnToMap Ontology, focused on concept “Place” .....	91

## Executive Summary

This document (D3.1) presents the architecture of the WeGovNow platform which is to be set up and piloted in three municipalities during the reminder of the project. It represents the first deliverable of the prototype development work strand (WP3) within the overall project. To our knowledge for the first time, WeGovNow aims at integrating a number of civic engagement applications that existed prior to the project together with various software components to be newly developed into a single online platform. With a view to facilitating further utilisation of the WeGovNow platform within different local contexts, the architecture is furthermore to enable modular configuration of individual implementation instances. Also, it is to be principally extendable with further applications which potentially may be desired to be added in future implementation instances. Such a multi-faceted set of requirements poses a number of challenges for the design of the WeGovNow platform architecture, in particular when it comes to:

- The structure and the configuration of the platform;
- The development cycle and design process;
- The integration of a diverse range of software components into the overall platform;
- The achievement of a coherent user experiences across all platform components, including the achievement of appropriate levels of accessibility and usability by a variety of envisaged user groups such as citizens with varying capacities and municipal staff.

The platform architecture described throughout this reports caters for these requirements in different respects, providing different types of features respectively:

- Distributed features provided by individual components or groups of components, keeping their own data and being served from different locations;
- Ubiquitous features, supporting the resilience of the platform toward single components, which may or may not be enabled within a specific instance of WeGovNow and which are managed by different teams;
- Asynchronous and scalable features preventing the creation of bottlenecks within the platform and enabling an information flow compatible with the workload of single components;
- Incremental adoption of solutions and features by single components, considering the existence of constraints and requirements stemming from the development cycles of the single components.

As a result, the platform architecture comprises a set of core features (the WeGovNow core) which are strictly required for running the WeGovNow as infrastructure, and a set of optional components that can be flexibly added by the provider of civic engagement services. The architecture thus enables a flexible configuration of a given implementation instances by combining principally available components in a modular manner, and of related cross-component features respectively. The WeGovNow core provides

fundamental features for running a WeGovNow instance (core features) and represents the basic platform infrastructure for realising the seamless integration of individual WeGovNow components:

- **Unified Authentication System:** It provides functionalities concerning user registration, also using mainstream social networks accounts, and single sign-on.
- **Application Discovery Service:** This is an API service providing the list and details of currently available components in a WeGovNow instance.
- **Style Service:** This an API service providing WeGovNow style sheets dynamically to the components, it is used in particular to retrieve the instance personalisations such as colors, fonts, etc.
- **NavigationBar:** Again, this is an API service providing the description or the HTML source of WeGovNow navigation bar, including the button tabs to the current available components and the reference to the user profile.
- **Centralised User Profile:** It provides a user profile datastore accessible to all registered WeGovNow components to store, retrieve and share user profile information, it consent to keep user information updated across the platform.
- **AccessibilityProfile:** This is a wizard to collect and edit cross-platform user's profile setups about accessibility.
- **Centralised Activity Logger:** It provides centralised data logging within the WeGovNow platform and data integration aimed at integrating the knowledge about users and about geographical data shared in the platform.
- **InputMap:** This is an embeddable web-map to collect spatial input (point based references) and references to existing entities in OntoMap.
- **AreaView:** Again, this embeddable web-map to visualise summary information extracted from OntoMap.

The WeGovNow core orchestrates a number of - principally extendable – modular components to be integrated into the overall platform for piloting purposes. This is achieved according to a set of jointly agreed integration approaches, including:

- **Deep linking,** interconnection based on URLs;
- **Data embedding:** inclusion of data from other applications;
- **View embedding:** inclusion of pre-rendered views generated by other applications;
- **Trigger actions:** use of functionalities provided by other applications via API.

Further key characteristics of the platform architecture are described throughout the current report, and the analytical work carried out in order to support design-related decision making within the project team as well. A number of internal reports which are annexed to the main document provide details in relation to the latter. Also, the report describes a set of common tools, datasets, patterns, libraries and documentations developed by the project team so far. These are not only intended to support further prototype development internal to the project. Also, it is hoped that these represent a useful resource for further platform extensions potentially desired beyond the project.

## 1 Introduction

This document (D3.1) presents the architecture of the WeGovNow platform which is to be set up and piloted in three municipalities during the remainder of the project. It represents the first deliverable of the prototype development works strand (WP3) within the overall project. To our knowledge for the first time, WeGovNow aims at integrating a number of civic engagement applications that existed prior to the project together with various software components to be newly developed into a single online platform. With a view to facilitating further utilisation of the WeGovNow platform within different local contexts, the architecture is furthermore to enable modular configuration of individual implementation instances. Also, it is to be principally extendable with further applications which potentially may be desired to be added in future implementation instances. Such a multi-faceted set of requirements poses a number of challenges for the design of the WeGovNow platform architecture, in particular when it comes to:

- The structure and the configuration of the platform;
- The development cycle and design process;
- The integration of a diverse range of software components into the overall platform;
- The achievement of a coherent user experiences across all platform components, including the achievement of appropriate levels of accessibility and usability by a variety of envisaged user groups such as citizens with varying capacities and municipal staff.

The remainder of this report describes in what way these challenges are addressed by the WeGovNow platform architecture. This starts with a description of a set of core components of the WeGovNow architecture (the WeGovNow core) in Chapter 2). This is followed by a description of a set of modular components which are going to be integrated within the overall platform for piloting purposes (Chapter 3). The report then also details in what way these are interlinked with each other (Chapter 4). Further to this, it is described in what way the platform architecture responds to integration requirements imposed by the various civic engagement applications to be integrated and by service related integration requirements imposed by organisational models and work processes prevailing at the part of the pilot municipalities (Chapter 5). Next, particular integration requirements stemming from the modular nature of the WeGovNow platform when it comes to performance testing and monitoring are discussed (Chapter 6). Also, the report describes a set of common tools, datasets, patterns, libraries and documentations developed by the project team so far (Chapter 7). These are not only intended to support further prototype development internal to the project. Also, it is hoped that these represent a useful resource for further platform extensions potentially desired beyond the project. Finally, key challenges faced when designing the architecture presented throughout this document and in what way these were addressed (Chapter 7).

The following descriptions of WeGovNow functionalities and components build upon the generic concepts presented in “conceptual framework D1.2”. To some extent

redundancies are therefore unavoidable to provide the reader with a self-standing document which details the initial concepts presented in D1.2.

## 2 WeGovNow overall architecture

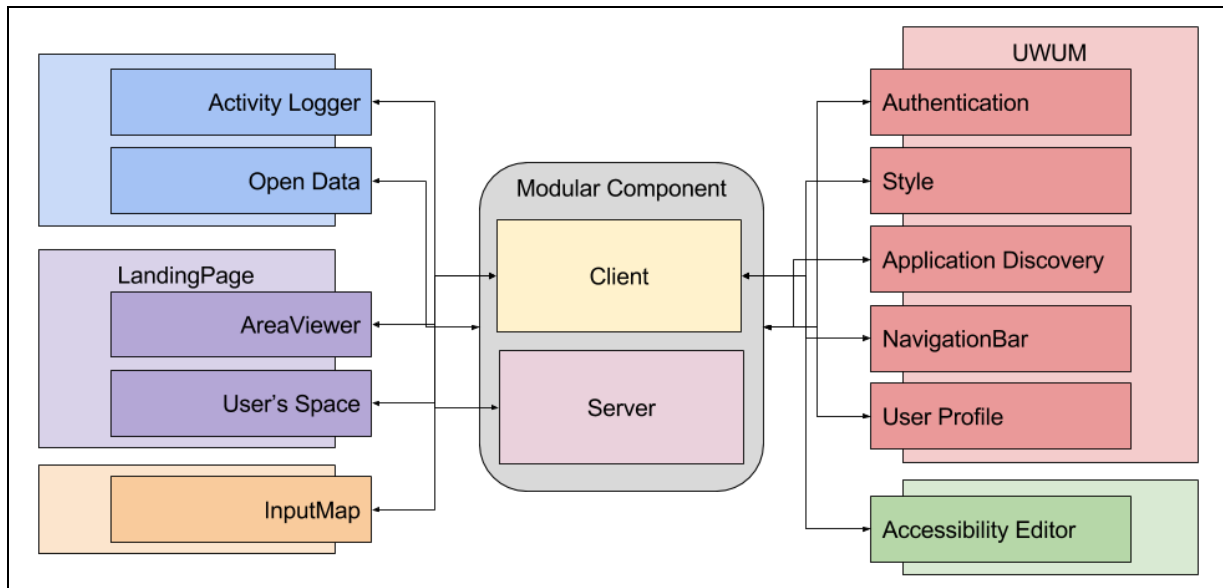
The WeGovNow platform comprises a set of core features (the WeGovNow core), which are strictly required to run the WeGovNow platform, and a set of modular components. A local WeGovNow implementation instance can thus be flexibly configured to enable any combination of the modular components and related cross-component features respectively.

The core features themselves are modular as well, thereby providing fundamental functionalities to run a local WeGovNow implementation instance. The WeGovNow core thus provides features (core features) that are essential for realising the seamless integration of individual WeGovNow components. It can briefly be summarised as follows:

- **Unified Authentication System:** It provides Functionalities concerning user registration, also using mainstream social networks accounts, and single sign-on.
- **Application Discovery Service:** This is an API service providing the list and details of currently available components in a WeGovNow instance.
- **Style Service:** This is an API service providing WeGovNow style sheets dynamically to the components, it is used in particular to retrieve the instance personalisation such as colours, fonts, etc.
- **NavigationBar:** Again, this is an API service providing the description or the HTML source of WeGovNow navigation bar, including the button tabs to the current available components and the reference to the user profile.
- **Centralised User Profile:** It provides a user profile data store accessible to all registered WeGovNow components to store, retrieve and share user profile information, it consent to keep user information updated across the platform.
- **AccessibilityProfile:** This is a wizard to collect and edit cross-platform user's profile setups about accessibility.
- **Centralised Activity Logger:** It provides centralised data logging within the WeGovNow platform and data integration aimed at integrating the knowledge about users and about geographical data shared in the platform.
- **InputMap:** This is an embeddable web map to collect spatial input (point based references) and references to existing entities in OntoMap.
- **AreaView:** embeddable web map to visualise summary information extracted from OntoMap.

In a local implementation instance, the WeGovNow core summarised above can be flexibly extended with modular components as part of the platform configuration process.

*Exhibit 1: WeGovNow core features provides the environment to integrate modular components accordingly to the specific needs of each WeGovNow instance*



Core features are used by modular components (Exhibit 1) to:

- Establish connections with other WeGovNow components
- Provide cross-component features
- Unify the appearance to the rest of a WeGovNow instance
- Synchronise consents

As any other modular component, core features themselves are optional as far as their in non-essential functionalities are concerned: if a core feature is not fully enabled in a particular implementation instance of WeGovNow, only its critical functionalities will be available. In the following subsections, the WeGovNow core is described in more detail.

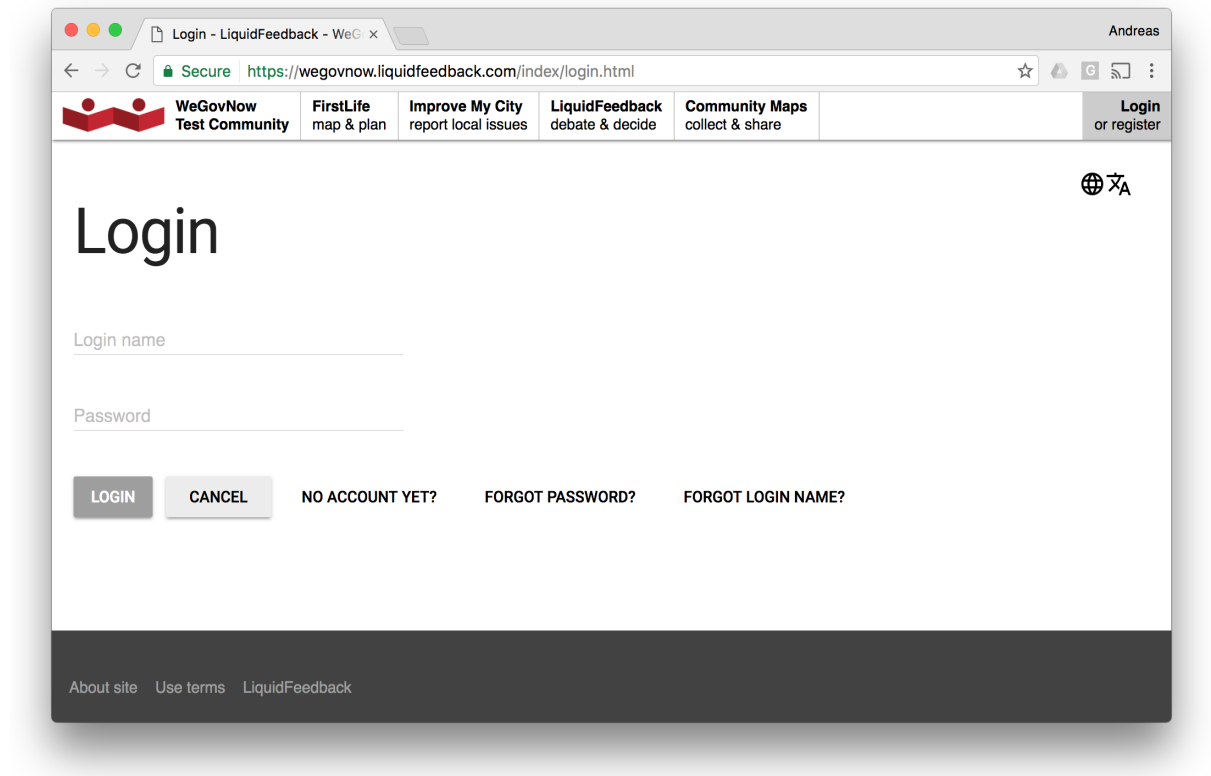
## 2.1 Unified WeGovNow Authentication System

The Unified WeGovNow User Management (UWUM) allows the distributed operation of the WeGovNow applications as well as third party applications (existing e-government applications and future applications) while the users experience the system as one seamless service. UWUM provides the backbone for a distributed system which is modular in terms of the used applications and the used identity providers.

The UWUM implementation is fully compliant with the proposed OAuth 2.0 standard (RFC 6749) while extending it (compared to any other known OAuth 2.0 implementation) in such way that additional components can be incorporated on a per-site basis (e.g. through the administration of a municipality) or on a per-user basis. The latter empowers the users to utilise a potentially unlimited number of software tools, each in a particular security scope provided by OAuth 2.0.

A first draft of UWUM was presented at a dedicated meeting kicking off the platform development work strand within the overall project (WP3) (see Appendix 2.1.1). The underlying idea relies on a single-sign-on (SSO) solution on OAuth 2.0's Authorization Code flow. It was further agreed that Transport Layer Security (TLS) is to be used to secure all communication between UWUM and other components.

*Exhibit 2: Login page delivered by UWUM.*



In addition to single-sign-on, UWUM's capabilities include:

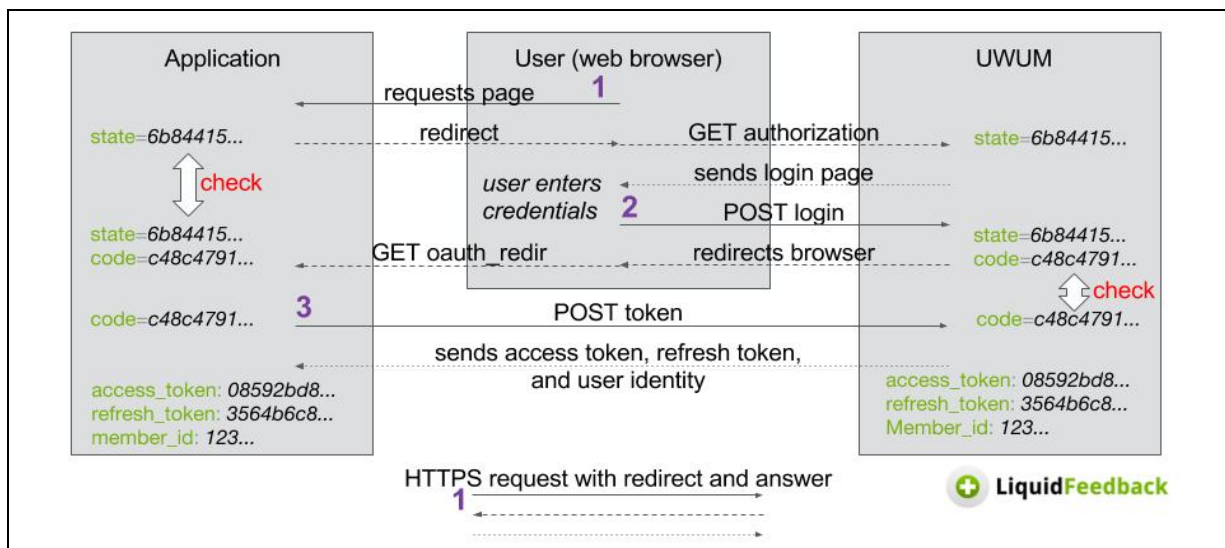
- Three additional API endpoints for integration (described in sections 2.2. - 2.4):
  - a "navigation" endpoint to incorporate a common WeGovNow navigation bar,
  - a "style" endpoint to retrieve style information (e.g. a color scheme), and
  - a "client" endpoint for application and service discovery,
- Additional support for the Implicit flow of OAuth 2.0 to allow easier authorization of lightweight JavaScript clients,
- Two different methods for dynamic client registration for yet-unforeseen 3rd-party tools:
  - through X.509 certificates (for client operators without DNS zone access),
  - through DNS TXT records (which may be easier for lightweight JavaScript clients),
- Asynchronous (background) login checks through Cross-Origin Resource Sharing (CORS),
- Additional security mechanisms (for details, refer to sections 2.9, 2.15, and 5.5 of the UWUM work report in Appendix 2.1.2).

UWUM thus allows WeGovNow to be a modular system that may be extended with different services (including those which are unknown at the time of development, due to “dynamic client registration”). UWUM is implemented by LiquidFeedback, thereby realising synergetic effects between the LiquidFeedback API and the new features required by UWUM. The remaining consortium partners have started to integrated their components in September 2016.

### 2.1.1 Secure authentication and authorization

For reasons of interoperability and security, WeGovNow aims to create an implementation that is fully compliant with the OAuth 2.0 Authorization Framework as described in RFC 6749, but extended in such way that it allows for secure user authentication following the OAuth 2.0 Authorization Code flow. Special security considerations were taken into account, for instance client identity verification (through X.509 certificates) to repel authorisation code substitution attacks. For further security considerations refer to sections 2.2 to 2.7, sections 2.9, 2.15, 2.17, 5.5, 5.7, 5.8, 5.10 of this deliverable and to section 5.14 of a dedicated “Work report on Unified WeGovNov User Management (UWUM) development” annexed to the current report (Appendix 2.1.2).

*Exhibit 3: OAuth 2.0 Authorization Code flow*



RFC 6749 defines several roles (“authorisation server”, “client”, and “resource server”). The UWUM component as implemented by LiquidFeedback takes the role of the “authorisation server”. Other WeGovNow components will take the role of “clients” but may also act as “resource server” for other components. This allows other components to interact with each other, while UWUM is responsible for user authentication and authorisation.

A detailed description of the UWUM development work done so far including challenges, design decisions and security considerations can be found in the above-mentioned “Work report on Unified WeGovNow User Management (UWUM) development” (Appendix 2.1.2).

### 2.1.2 Dynamic client registration

UWUM provides two methods of client registration:

- registering clients through the municipality (or their technical administration) or an organisation running a particular installation of WeGovNow,
- registration of any other (“dynamic”) client on a per-user basis by each user who wishes to use that client to access WeGovNow (machine accessibility).

Manual client registration by the municipality is only suitable for those clients that are known at the time of deployment, or for a service-centred approach where a given software provides only a single service (e.g. Facebook, Google, Twitter, etc). An open source solution, however, could be installed at several sites (e.g. different municipalities) by different service providers, and future (unforeseen) applications could be developed. It is therefore not meaningful for these applications being required to be registered by a particular municipality. Instead, the development of generic 3rd-party applications should be enabled, where each application is automatically usable for any installation of WeGovNow that uses the UWUM server software. Of course, access can be limited by each user and by the security configuration of each UWUM server.

To allow for a secure 3rd-party application access, LiquidFeedback implemented a dynamic client registration protocol that keeps implementation complexity at a minimum while providing good security properties which outperform many other solutions for client registration due to requiring direct access to the DNS zone of the domain (for adding a TXT record) or credentials in form of a publicly trusted TLS certificate with corresponding key (only accessible by the domain owner). Details on dynamic client registration are provided in subsection 2.4.2 of the above-mentioned “Work report on Unified WeGovNow User Management (UWUM) development” (Appendix 2.1.2).

## 2.2 Application Discovery Service

The client endpoint of the UWUM service is supposed to return a list of all system applications and, if an access token is provided, a list of all registered dynamic clients for the corresponding user. This facilitates communication between applications that have not been known at the time of development. Specification and implementation of this endpoint will be done by LiquidFeedback in close cooperation with UniTo (see also Appendix 2.1.2).

## 2.3 Style Service

A dedicated style endpoint of the UWUM service provides basic colour definitions for a primary and an accent colour as 24-bit RGB triplet to be able to customize the unified visual look of all WeGovNow applications for a particular installation by central configuration. Additional colours can be derived from these two base colours. If the UWUM server gets configured with colours from the Material Design colour palette, the corresponding Material Design colour name of the primary and the accent colour is also provided (see also Appendix 2.1.2).

## 2.4 NavigationBar

In order to integrate all WeGovNow applications in such way that they look and feel like a single application, all WeGovNow applications share a common WeGovNow navigation bar (see Appendix 2.4). The navigation endpoint of the UWUM server returns this navigation bar to be included by each WeGovNow application. This way, modifications to the navigation bar can be made at a central place without the need to change every single application. Either a login button or the user name with a link to a user page (where logout is possible) is included in the navigation bar, depending on whether an access token is provided when calling the endpoint. A detailed technical description of the NavigationBar can be found in sections 3.1, 5.10, 5.11, 5.12 of Appendix 2.1.2. Notes on the look-and-feel are provided in Appendix 2.4.

## 2.5 Centralised User Profile

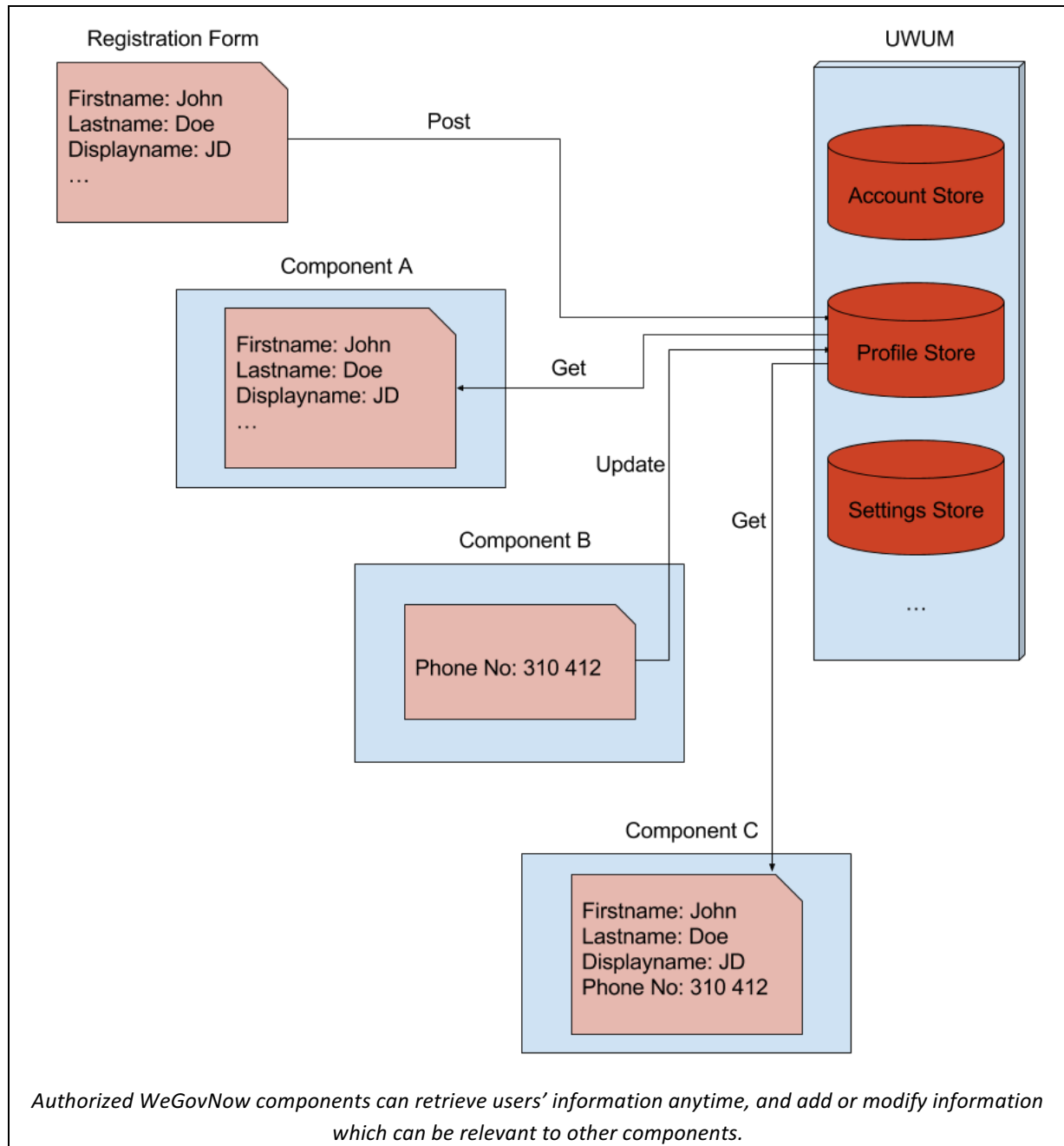
The WeGovNow platform provides a centralised user profile repository to its components where user profile information of common use among the platform are collected. The centralised user profile is an UWUM component, it stores the information acquired during the registration process, and can be extended and modified anytime by all authorized WeGovNow components (Exhibit 4).

The centralised user profile has the following main functions:

- Single synchronisation point for common user information (such as firstname, lastname, displayname, email, etc.) avoiding the multiple request of user profile data;
- Propagating the updates of users' profile across WeGovNow platform (Exhibit 4) avoiding inconsistent information;
- Increasing the integration of WeGovNow components providing a mechanism to share user's information platform wide.

The user profile data fields and their type will be configurable per installation. These data fields can be of standard types such as text, number, image, location or complex JSON data fields. Standard type fields will be automatically editable by UWUM's built in editor but can also be updated using the API. Complex JSON data fields need to be updated using the API by the corresponding component(s) using such fields.

*Exhibit 4: UWUM registration form initialises the user profile with the form data entered during registration*



Using the API, the Trusted Marketplace component will also provide a feature to change the content of the centralised user profile at any time. This will include a dialogue (wizard) guiding the users through user profile fields.

## 2.6 Centralised Settings

The WeGovNow platform provides a centralised settings management to its components where user related settings of common use among the platform are collected. The

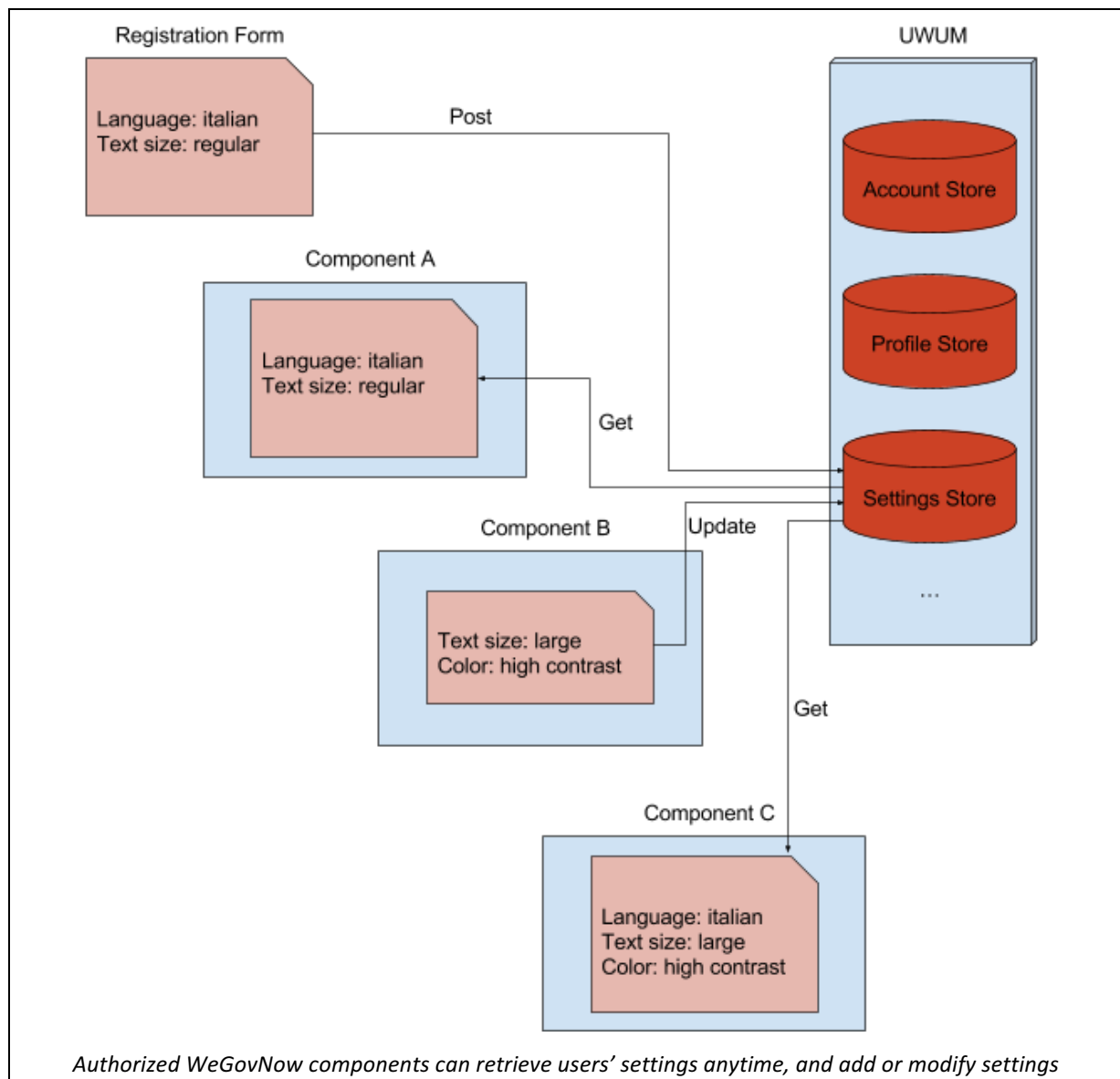
centralised settings store is an UWUM component, it stores the information acquired during the registration process, and it can be extended and modified anytime by all authorized WeGovNow components (Exhibit 4).

### 2.6.1 Accessibility Settings

Basic support for accessibility is extended from specific features of components to the overall WeGovNow platform. Specifically, users' preferences about text size, contrast, languages are collected once and shared among WeGovNow components through UWUM's user settings.

User's preferences are collected through a wizard interface meant to guide users in defining their best condition of use. The update of preferences is propagated via UWUM, editing the user settings (Exhibit 5).

*Exhibit 5: UWUM registration form initialises the user settings*



WeGovNow components can retrieve user's preferences from UWUM and adapt their interface at runtime, without requiring further setup from the user.

The collection of user's preferences and setups is being initialised during the registration step and enriched later on by the different components. The access to the user's preferences is available platform-wise via the navigation bar, the page setup presents the status of the user profile in UWUM and the direct link to each components setup page.

## 2.7 Centralised Activity Logger

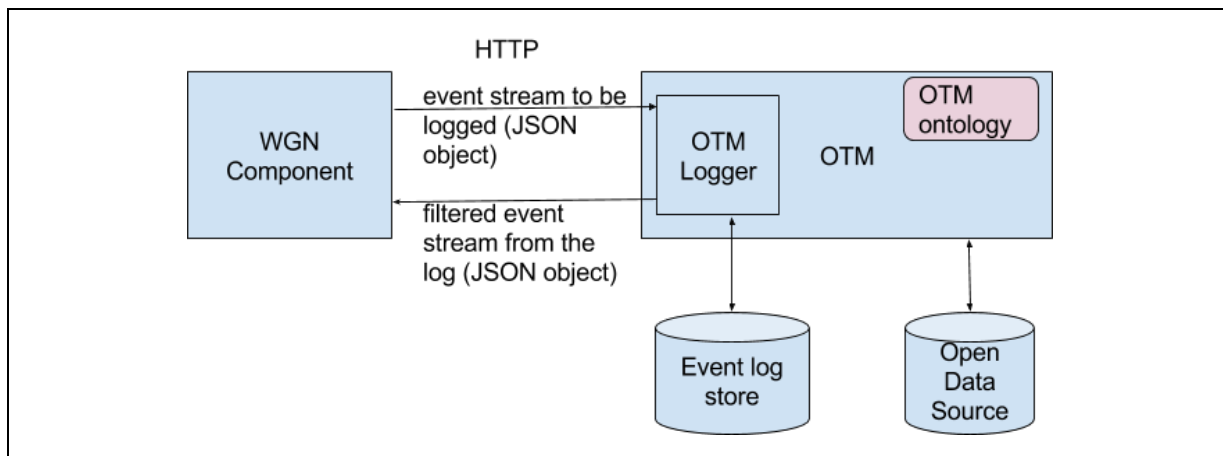
The following description presents the OnToMap Logger. The OnToMap Logger has two primary roles in the WeGovNow platform:

- *Centralised logging of user activities within the WeGovNow platform.* The OnToMap Logger is a centralised collector of the history of actions performed by WeGovNow users by interacting with the WeGovNow applications. The goal is that of achieving a unified perspective on user behaviour, e.g., to support cross-application personalization. Within the WeGovNow platform, applications push their log data to a single component (the OnToMap Logger), which merges the information managing a unified history of user activities. In order to guarantee control over private data, we assume that WeGovNow applications only push to the Logger the data they are willing to share in the WeGovNow platform. The Logger is thus unaware of, and cannot distribute, the user activity information that applications want to keep private. In order to effectively support information retrieval, centralised logging has to be coupled with data integration. In fact, as applications can adopt different terminologies, the user activities they log might seem to refer to different types of information even when it is not the case. This leads to the second role of the Logger, described in the following.
- *Data integration across WeGovNow applications.* The Logger provides a unified view on the data shared in the WeGovNow platform, including the Open Data managed by OnToMap, in order to enable WeGovNow applications to retrieve information collected about geographical objects, initiatives, issues, and so forth. Two main challenges have to be addressed:
  - As applications can use diverse conceptual models for representing geographical information, heterogeneous data descriptions have to be reconciled. The OnToMap Logger addresses this issue by exploiting the semantic representation of geographical information provided by OnToMap Ontology, that is used as an interlingua among WeGovNow applications (see also section 4.12). The ontology formally represents the concepts in which the data shared among the WeGovNow applications can be classified: the data categories defined by an application can be mapped to the concepts of the ontology, at application integration time, to define the rules for translating the information provided by the application to the common ontology format (see also section 6.3).

- The second challenge is the identification of the geographical objects described by the WeGovNow applications, which might refer to the same entity in distinct ways; e.g., two applications might provide different URIs for the same school in Torino. The OnToMap Logger addresses this issue by managing a registry that unifies the URIs of the geographical entities managed in different WeGovNow applications, i.e. by recognizing their identity (see also section 3.6).

### 2.7.1 OnToMap Logger architecture and APIs

*Exhibit 6: Interaction between a WeGovNow (WGN) component, and the OnToMap (OTM) Logger and ontology*



The OnToMap Logger enables WeGovNow applications to push streams of events to be logged, and to retrieve filtered log information; e.g., the activities performed by a certain user in all the WeGovNow applications on a certain date. At an abstract level, the logging of user activities is carried out as follows (see the figure above):

- It acquires events to be logged that are described either in the terminology used by the WeGovNow applications, or in the OnToMap Ontology format. For each event *E* pushed by a WeGovNow application, the Logger translates the arguments of *E* to the terminology provided by the OnToMap Ontology (if needed), in order to maintain a single representation of the geographical entities manipulated in the logged activity. Then, the Logger stores the translated events into the Event Log store. This translation makes it possible to track user activities and to retrieve geographical information crowdsourced by the WeGovNow applications using a unified dictionary.
- When invoked to retrieve logged data, the Logger returns events expressed in OnToMap terminology.

For instance, let's suppose that a WeGovNow Application 1 event reports the fact that an UWUM user has updated a geographical object: the object describes a kindergarten named "Asilo\_1", with address ("indirizzo") and phone number ("telefono"). In the terminology of Application 1, kindergartens are represented by concept "ChildCare", mapped to OnToMap Ontology concept "Kindergarten". The table below shows the event pushed to the Logger,

(described in Application 1 terminology), and the translated one, to be saved into the Event Log Store.

*Exhibit 7: Translation of a user activity event to the format managed by the OnToMap Logger*

Event expressed in the terminology adopted by Application 1	Event expressed in OnToMap Logger format
<pre>{   "actor": 123456,   "timestamp": 1234567,   "activity_type": "object_updated",   "activity_objects": [     {       "type": "Feature",       "properties": {         "hasType": "ChildCare",         "external_url":           "http://app1/Asilo1"         "denominazione": "Asilo_1",         "indirizzo": "Indirizzo_1",         "telefono": "0123456",         "retta_mensile": 100       },       "geometry": null     }   ] }</pre>	<pre>{   "actor": 123456,   "timestamp": 1234567,   "activity_type": "object_updated",   "activity_objects": [     {       "type": "Feature",       "properties": {         "hasType": "Kindergarten",         "external_url":           "http://app1/Asilo1",         "hasName": "Asilo_1",         "hasAddress": "Indirizzo_1",         "hasPhoneNumber": "0123456",         "hasMonthlyRate": {           "value": 100,           "unit": "EUR"         }       },       "geometry": null     }   ] }</pre>

Notice that, as the events to be logged store a detailed description of the objects of the user actions (e.g., geographical entities created/modified by users, and their properties), their memorisation supports a centralised and unified storage of geographical information in the OnToMap Logger, supporting cross-application data retrieval.

Specifically, the OnToMap Logger offers the following APIs (specification available at <https://ontomap.eu>):

- An API to receive a continuous stream of event descriptions from WeGovNow applications. Events can be pushed synchronously (one by one, as soon as the user

actions happen), or asynchronously (pushing a list of events to be stored). The action descriptions, represented as JSON objects, specify user activities on data items; e.g., creating, updating, removing, commenting some data objects. They can be expressed in the terminology of the WeGovNow application (to be translated to OnToMap format), or directly in the OnToMap format. In particular:

- The event descriptions can include the specifications of the geographical objects manipulated by the users.
- The specification of each geographical object must be a GeoJSON “Feature”<sup>1</sup> and it must include two mandatory properties:
  - The “external\_url” property to specify the deep link to the geographical object described by the Feature. This is aimed at supporting direct access to the object in its source application.
  - The “hasType” property, needed to declare the category/concept of the object. The reason is the fact that a JSON object cannot be queried to retrieve the category/concept to which it belongs, unless this information is specified in the object itself.

The Logger assumes that the user IDs occurring in the pushed events are valid; i.e., users have been authenticated by the WeGovNow applications before pushing the activity information.

The schema in [https://ontomap.eu/logging\\_schema.json](https://ontomap.eu/logging_schema.json) defines the structure of the events accepted by the Logger and Exhibit 8 below describes this structure in natural language for the reader’s convenience. Exhibit 7 shows an example user activity event pushed to the OnToMap Logger and its translation to the Logger format.

- A set of APIs to retrieve subsets of the information maintained by the OnToMap Logger:
  - The APIs support event filtering; e.g., by time interval, geographical bounding box, user, geographical object, issue, initiative, type of activity, source application.
  - The invocation of an API returns a JSON object storing the filtered data. The JSON object is formatted, as above, according to the schema in [https://ontomap.eu/logging\\_schema.json](https://ontomap.eu/logging_schema.json) (see Exhibit 8, as well). The geographical objects occurring within the returned data are represented as GeoJSON “Feature” (with “hasType” and “external-url” attributes) and refer to the concepts of the OnToMap Ontology.

Only authorized WeGovNow applications must be able to use the Logger APIs for storing and retrieving data. Thus, the OnToMap Logger requires that every request includes a X.509 client certificate signed by LiquidFeedback.

---

<sup>1</sup> See <http://geojson.org/geojson-spec.html#feature-objects>

The following table describes in natural language the structure of the user activity events accepted by the OnToMap Logger JSON schema specifying the structure of the events accepted by the OnToMap Logger.

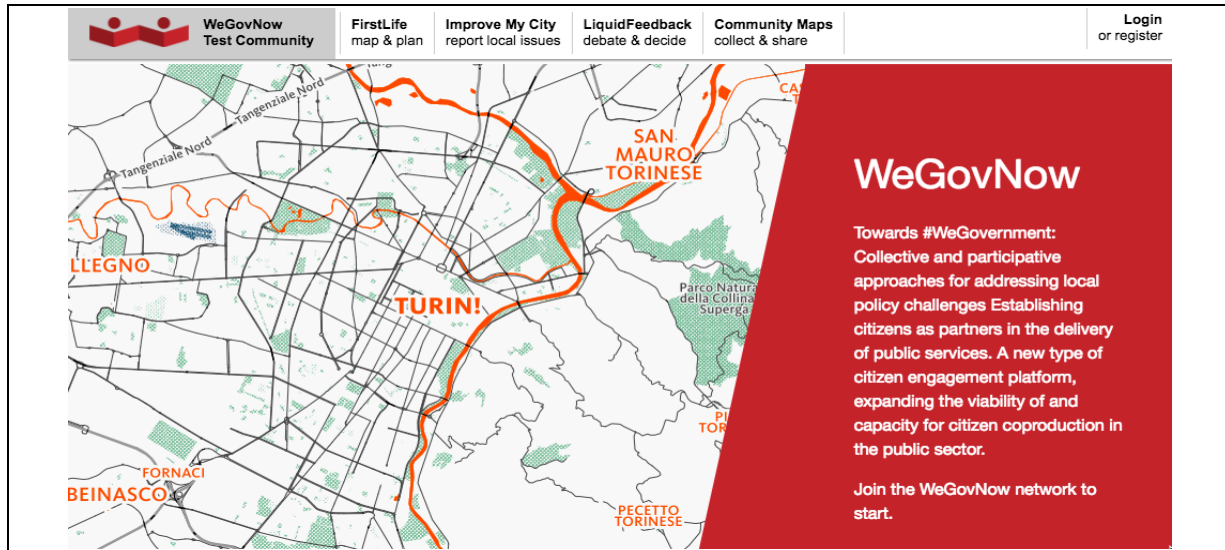
*Exhibit 8: Structure of the user activity events accepted by the OnToMap Logger*

Attribute	Description
actor (integer)	<b>Required.</b> The UWUM ID of the user who performed the action to be logged. We assume that the user has been correctly authenticated in the frontend application that invoked the logger.
timestamp (integer)	<b>Required.</b> The timestamp of the action to be logged, represented as a Unix timestamp in milliseconds.
activity_type (string)	<b>Required.</b> A string representing the type of action performed by the user that has to be logged.
application (string)	A string containing the domain name of the application in which the user performed the logged action. Please note that this field is optional and it is not required when pushing an event to the OnToMap Logger, because this information is extrapolated from the certificate used for sending the request. However, the Logger adds this information to the event structure in order to be able to include a reference to the source application when returning event streams from the log.
activity_objects (Array)	A JSON Array containing items of type <a href="#">GeoJSON Feature</a> , representing the object(s) of the action performed by the user. If an action object is not a geographical item (e.g Issues, Initiatives etc.) the “geometry” field of the Feature can be set to null. The “properties” field of each Feature must contain a field “hasType” representing the category of the item and a “external_url” field representing the deep link to the item.
geo_objects (Array)	A JSON Array of objects, each representing a geographical object to which the action performed by the user relates. This field makes sense only when the geographical object is not the direct object of the action performed by the user. E.g. if a user edits an issue, that is related to a geographical item managed by a WeGovNow application, geo_objects includes a reference to that geographical item. Each object contained in “geo_objects” must include: <ul style="list-style-type: none"> <li>• an “external_url” field representing the deep link to the item, and</li> <li>• an “application” field representing the domain name of the application managing the geographical item.</li> </ul>
details (object)	A JSON object containing application and action-specific details about the action performed by the user; e.g., changes to the user profile on a WeGovNow application might be stored here. This field is stored as provided by the application.

## 2.8 Landing page

The LandingPage (Exhibit 9) is the component handling the first access by users to the WeGovNow platform. It provides pointers to project information in the WeGovNow website, an overview of the current status of WeGovNow and a dynamic toolbar guiding users through login/registration, to check the status of their contents and the like.

*Exhibit 9: LandingPage screenshot*

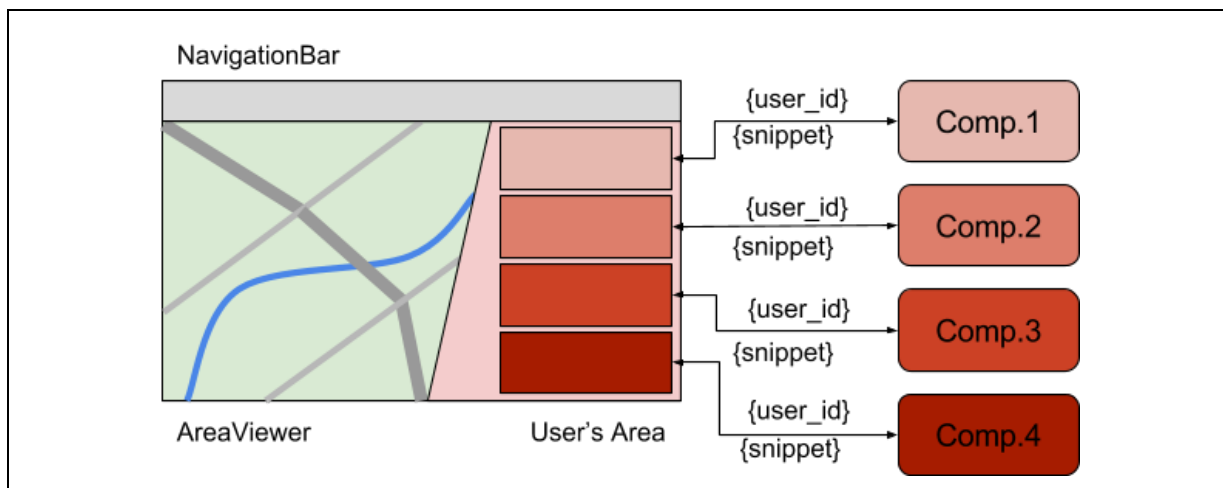


The LandingPage is a web application based on Angular 2.x integrating an AreaViewer component and the WeGovNow NavigationBar.

### 2.8.1 Structure

The LandingPage structure hosts two sections: a viewer of the instance area (trial site) and a modular user area that can be personalised in accordance with the WeGovNow instance parameters (Exhibit 10). The User area is meant to host snippet about user's activities from the WeGovNow components.

Exhibit 10: LandingPage structure



### 2.8.2 User's area

The user's area can host modules from WeGovNow components, in order to expose users' updates and fast access to component features. It is generated from the current configuration of WeGovNow: available components and general configuration.

The LandingPage will provide a default module for the user's area providing direct links to user management features. Additional modules will be developed according to the particular requirements coming from municipalities and local stakeholders.

The module in the user area is developed according to a Angular 2.x component pattern<sup>2</sup>.

### 2.8.3 AreaViewer

The AreaViewer is a LandingPage module providing a summary visualisation of the WeGovNow instance's current status based on the map. It is an independent module that can be included in other WeGovNow components.

### 2.8.4 Authentication support

The LandingPage handles login/registration callbacks from UWUM. It includes callback pages to display error and success messages from UWUM authentication service. The LandingPage provides a single point to manage cross platform errors, and a contact point for seeking assistance.

## 2.9 InputMap

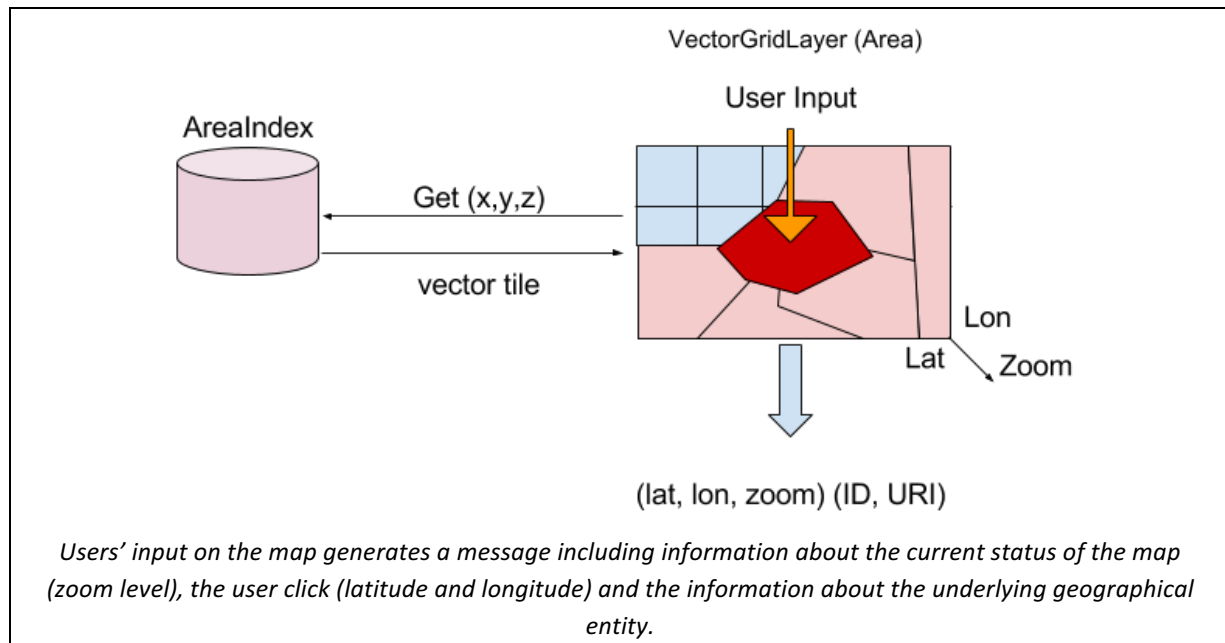
The InputMap provides a unified system for spatial input in WeGovNow. It is meant to enhance the overall look and feel of the WeGovNow platform, providing an alternative to the different input methods used within WeGovNow components. Moreover, the InputMap, working in combination with OnToMap, has the role to consolidate the data

<sup>2</sup> <https://angular.io/docs/ts/latest/guide/architecture.html>

within WeGovNow, providing a mechanism to create explicit connections between components' data and entities in OnToMap.

The InputMap is web map based on Leaflet  $\geq 1$  and Angular  $\geq 2$  to collect enhanced spatial information as an input on a map. The map output is a triple (x, y, z) equivalent to latitude, longitude and map zoom level. Moreover, the output indicates the ID and URI of the relevant geographical entity if available.

Exhibit 11: Input Map



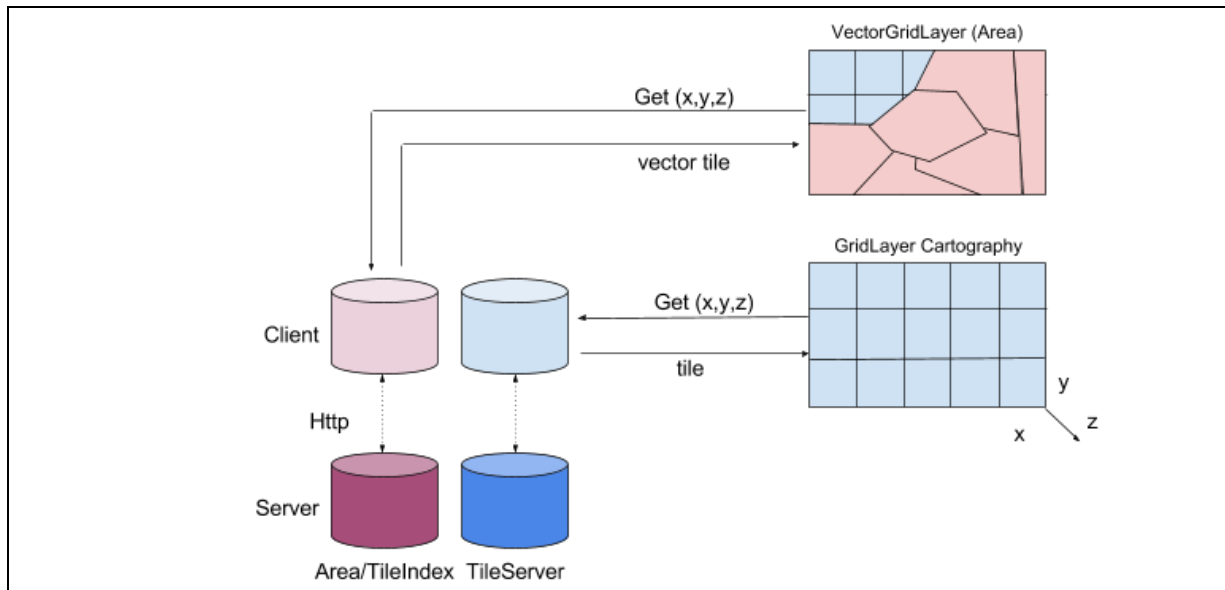
### 2.9.1 Use

The InputMap is based on Leaflet VectorGrid layer, an extension of Leaflet 1.x<sup>3</sup>. The Leaflet map is scaffold by a web application in Angular 2.x to manage the communication between the web map and the data source for caching purposes and to propagate the events to the agent (browser) using HTML5 window.postMessage function<sup>4</sup>.

<sup>3</sup> <https://github.com/Leaflet/Leaflet.VectorGrid>

<sup>4</sup> <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>.

*Exhibit 12: InputMap constitutes of a cartographic layer and a vector layer of relevant geographical entities*



The InputMap can be included as embed (iFrame) in any web application as a substitute for other input maps such as Google Maps or Leaflet. As embed it does not require code integration aside from a listener for the window.postMessage call. Since the output of the InputMap is an extension of the output of common input on maps it is completely alternative to other maps.

### 2.9.2 Data sources

The InputMap uses two data sources: a tile server to retrieve the cartography and an Area/TileIndex mapping geographical entities with tiles. The geographical entities within the Area/TileIndex refers to available open data and other geographical sources.

### 2.9.3 TileServer

Sources of cartographic information are indexed as tiles (x, y, z) with the format of SVG, GeoJSON or image (PNG, JPEG). The source can be locally stored or remotely retrieved (handled by the local storage).

### 2.9.4 Area/TileIndex

The source of geographical information about an area is available in GeoJSON format. The source is indexed via ID, it is stored locally or remotely (handled by the local storage component). It is a feature collection of the overall shape of the area and the included elements, providing a summary (aggregate information) based on the users' activities in the platform.

## 2.10 AreaViewer

The AreaViewer is a web map based application providing a view of WeGovNow aggregated data based on OnToMap. It works in combination with the InputMap component, exploiting the explicit relations between application data of WeGovNow components and OnToMap entities. The AreaViewer is a component of the LandingPage that can be included in any WeGovNow component through iFrame (embed) and be controlled via url. The purposes of the AreaViewer are enhancing the overall look and feel of WeGovNow platform and providing a coherent visualisation of the current status of WeGovNow instances across components. It is web map based on Leaflet 1x. and Angular 2.x.

### 2.10.1 Structure and use of AreaViewer

The AreaViewer is based on a map viewer (Leaflet 1.x) including three layers:

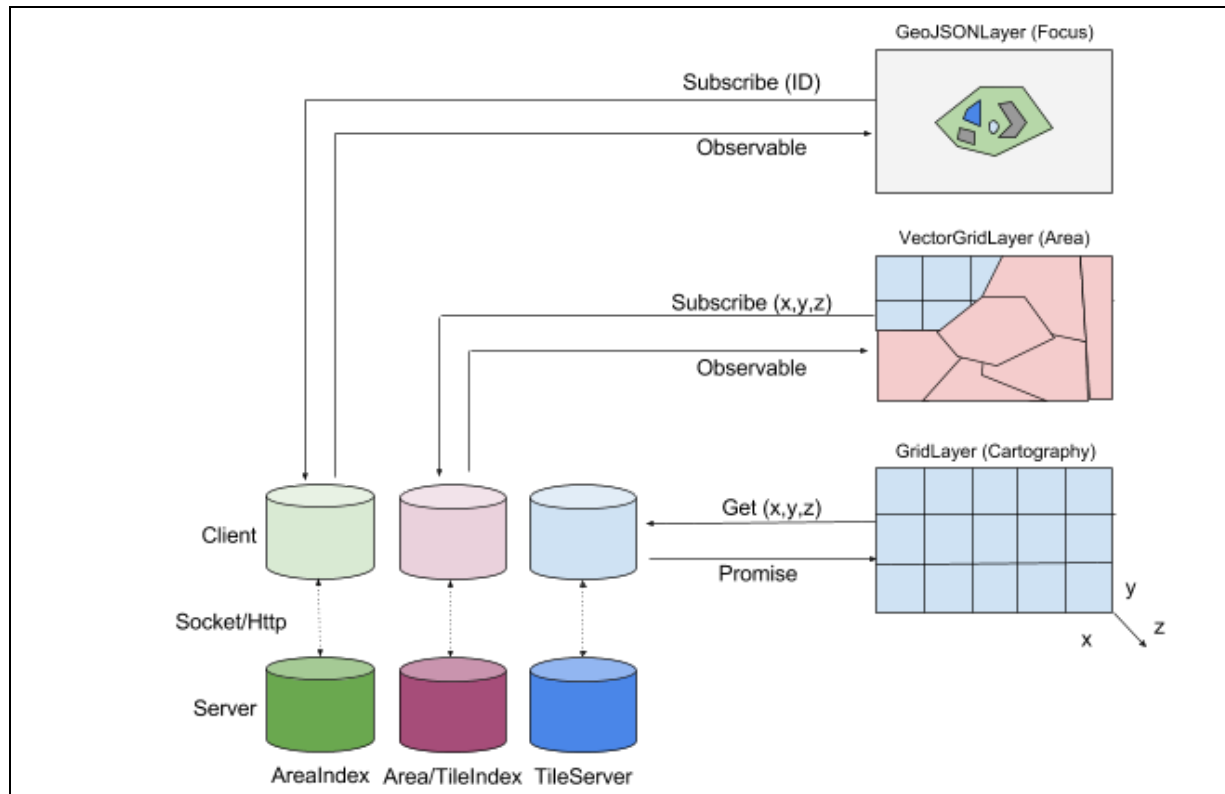
- Cartographic layer (GridLayer) providing the map base;
- Area layer (vectorGridLayer) including the relevant geographical entities for the current status of the map (bounding box and zoom level);
- Focus layer (GeoJSONLayer) representing the detail of a specific area.

The AreaViewer provides users the possibility to explore a map (cartography) enhanced with a summary of the current activities retrieved from OnToMap (area layer). When selecting a specific area, users can explore contents increasing the level of details (focus layer).

Summary information used to build both, area and focus layers, are precalculated exploiting the aggregation of WeGovNow application data on OnToMap's entities. The area descriptions and area summaries are accessed through a web service and cached within the agent (browser).

The AreaViewer can be embedded as iFrame and controlled via url parameters by the hosting applications. For instance, a WeGovNow component can focus the viewer adding a GET parameter "entity={entity\_id}" or change the map status (lat={latitude},lng={longitude},z={zoom}). The use of the AreaViewer does not require extra development aside the management of the url parameters.

*Exhibit 13: AreaViewer is composed by three main layers connected to three data sources specifically designed to reduce the computational costs at client side, and cacheable*



### 2.10.2 Notifications

The AreaViewer can be used to display notification on the map. The notification system is based on the use of live database for the AreaIndex and the Area/TileIndex. The live database will push a change notification at the client level triggering the update of the relative view (layer): the change within the Area/TileIndex triggers the update of the vectorGridLayer, and the AreaIndex triggers the update of the GeoJSONLayer.

### 2.10.3 Data Sources

The AreaViewer requires three data sources specifically designed to lower the computational cost of rendering and interacting with the data at client level. The data sources are cached at client level in order to reduce the network load and enhance the overall performances.

In order to support the use of the AreaViewer for displaying notifications the AreaIndex and Area/TileIndex data sources are implemented as a live database<sup>5</sup>. As a consequence, the communication client/server protocol is based on websockets<sup>6</sup> to seamlessly retrieve database slices and databases in real time.

<sup>5</sup> [https://en.wikipedia.org/wiki/Real-time\\_database](https://en.wikipedia.org/wiki/Real-time_database)

<sup>6</sup> [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

**Area:** The collection of the description of the geographical entities, with the contained entities and summaries. Areas are accessible via ID.

**Area/TileIndex:** The collection of an area with summary information indexed according to its containing tile (x,y,z).

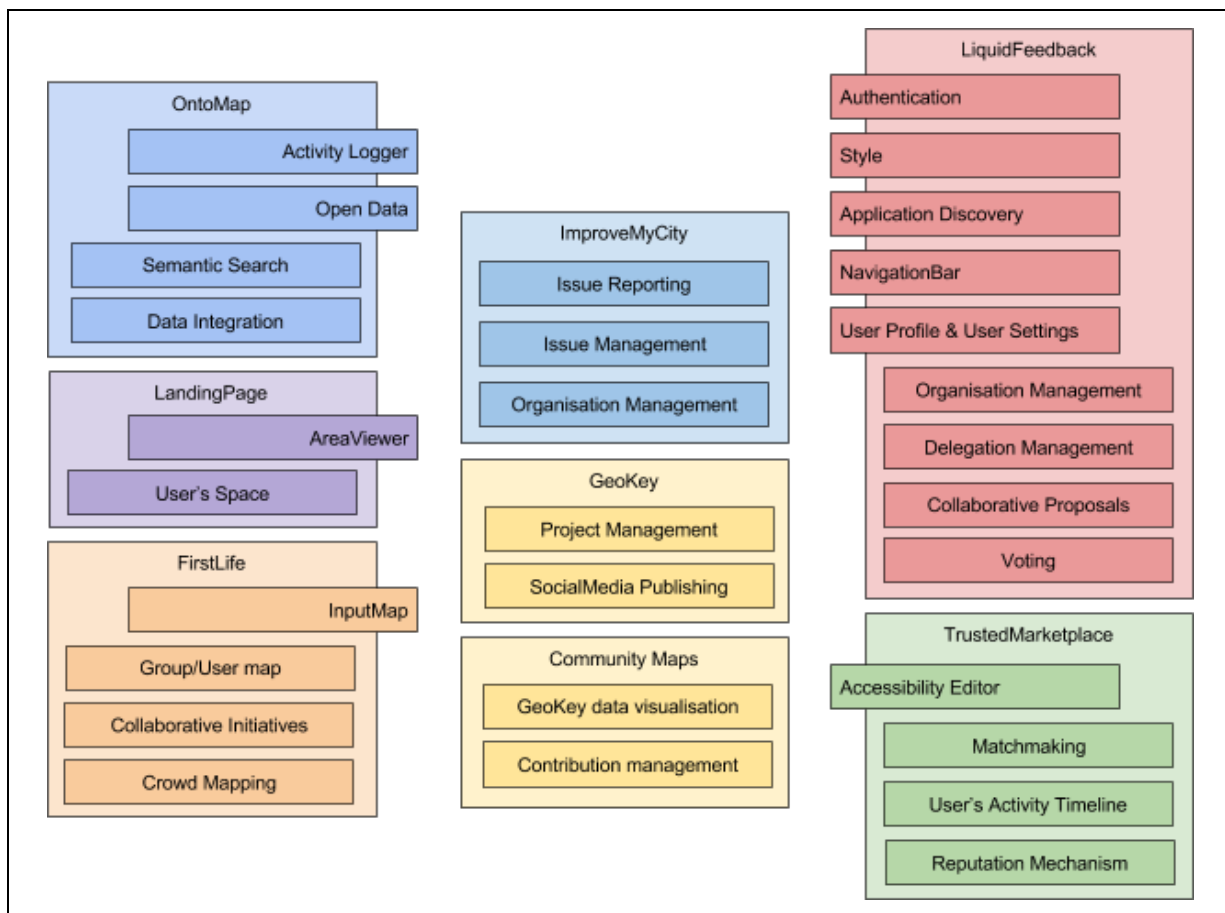
**TileServer:** Collection of cartographic sections indexed according to the tile grid<sup>7</sup>.

### 3 WeGovNow components

The WeGovNow platform is a modular web-based application including several modules (modular components) specifically extended to work in coordination as a unique integrated platform. WeGovNow is not limited to the existing modules but it is ready to be extended by registering new modular components (see also section 6.3).

The components which currently integrated into the WeGovNow platform (Exhibit 14) are existing civic engagement applications that were partly developed prior to the WeGovNow project. Together, they address a range of issues relevant in this context.

*Exhibit 14: WeGovNow components are interconnected through API and shared functionalities specifically developed to support a distributed orchestration of features*



<sup>7</sup> [https://wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames).

In technological regard, the current WeGovNow components support and use the core features of WeGovNow described earlier in this document (Chapter 2). WeGovNow adopts a set of solutions to realise a standards-based, modular environment that can be easily personalised according to local implementation requirements of each municipality. In particular, current and future WeGovNow components must (see also section 6.3):

- support OAuth 2.0 protocol;
- adopt UWUM (Unified WeGovNow User Management system) for authentication;
- send user's log activities to the Centralised Activity Logger;
- and include the NavigationBar.

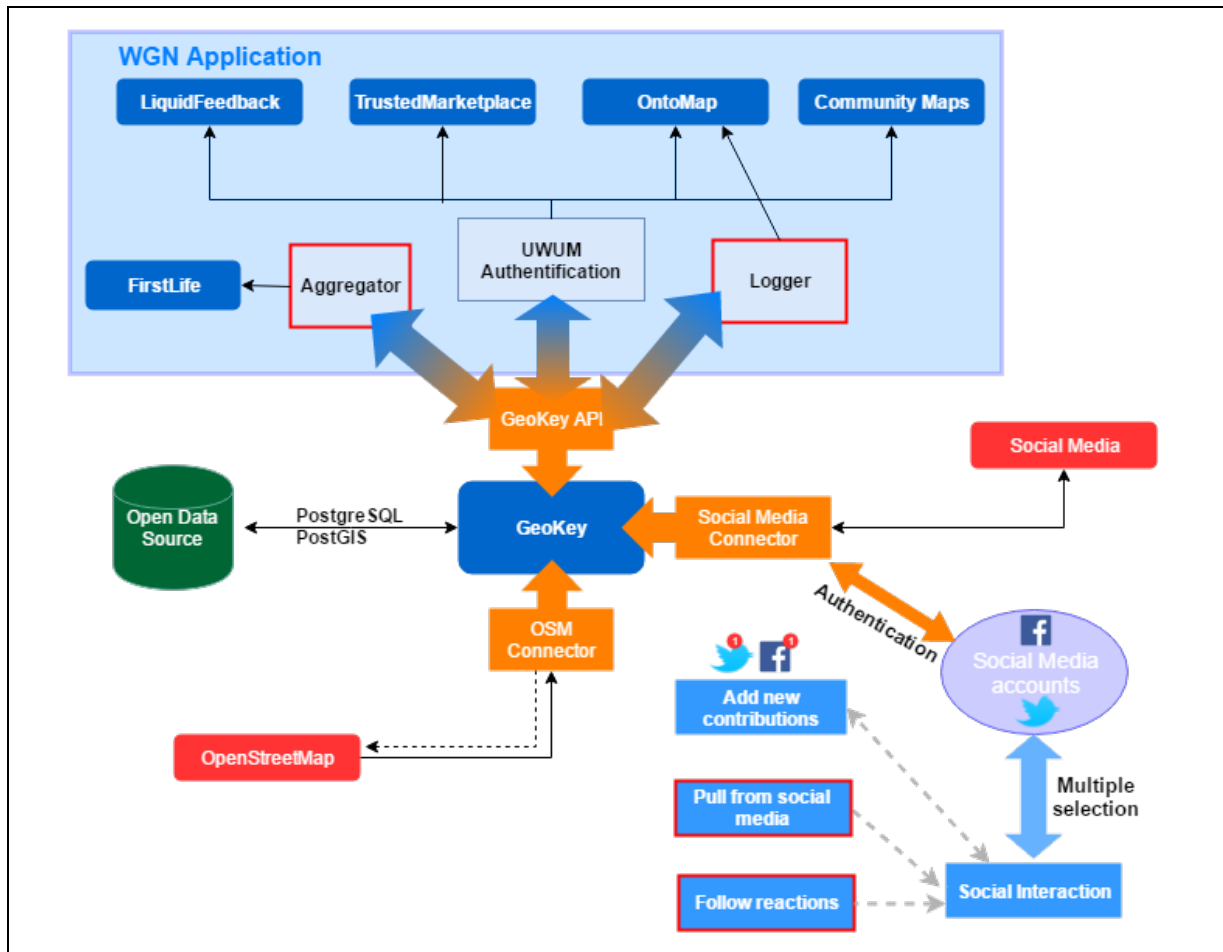
Moreover, there are a number of optional but strongly recommended requirements. In particular it is strongly recommended that future components:

- are WCAG 2.0 level AA compliant;
- follow the WeGovNow design guide and Material Design guidelines
- register the application on UWUM
- supporting the global accessibility setups
- rely on UWUM to manage and keep in synch the common field of user profiles
- replace with InputMap other maps used to get point-based location input
- replace common web maps by the AreaViewer

Along with the support to WeGovNow core, the current WeGovNow components are being further developed to accommodate new features, thereby exploiting their integration into the WeGovNow platform. In the following subsections, the current WeGovNow components are described in more detail.

### 3.1 GeoKey

Exhibit 15: GeoKey architecture and main components



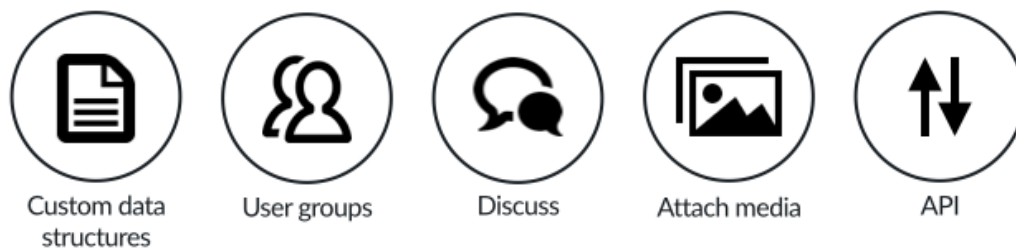
GeoKey is a web based platform for participatory mapping. GeoKey is the connecting point between data collection on the one hand and data utilisation through the analysis and visualization on the other hand.

The GeoKey architecture provides a database-driven backend storage together with a customer API. The structure of the database can be defined by the web-based administration tool which provides a user interface to set up and manage community mapping projects where users or administrators can create projects and define categories and the structure for the database itself. The API allows two main tasks namely interaction with data (data creation, editing, deleting) and the creation of projects which group data together. Apart from that the public REST API provides access to the system to third-party application to manage the data using the HTTP methods GET, POST and DELETE. By default this information is exchanged using JSON and there are existing extensions which allow users to export the data in XML or CSV allowing integration with GIS.

GeoKey provides storage facilities for user-generated geographic information and associated data such as documents, images, video and comments. It is a server-side application that supports structures for data collection and data validation according to these structures as well as managing user permissions. GeoKey allows adding extensions

that provide bespoke functionality to specific use cases, for instance special API end points added to and consumed with mobile client applications. This extensibility forms a core component of the integration process with WeGovNow.

*Exhibit 16: GeoKey's key features*



### 3.1.1 Extending GeoKey for WeGovNow

GeoKey uses django-allauth<sup>8</sup> library to allow users to authenticate via various social providers (for example, Twitter or Facebook). A custom django-allauth-uwum provider<sup>9</sup> was created to extend the native behaviour of the library for supporting the UWUM. X.509 certificate used for a secure authentication.

The default GeoKey behaviour is being extended for specific WeGovNow use cases using the custom geokey-wegovnow extension<sup>10</sup>. It enables such features as:

- Users are allowed to use GeoKey (and also Community Maps) with UWUM accounts only; Before any user can register to GeoKey (and also Community Maps) providing email but this configuration has been modified allowing only users who have been previously registered and authenticated with UWUM. If the users is not registered yet is automatically redirected to UWUM registration site.
- Display name and email address of each user are automatically updated when changed on the UWUM account;
- Users are able to reach the UWUM settings from within the GeoKey interface;
- Users are automatically logged out of GeoKey (and also Community Maps) when they log out of UWUM;
- WeGovNow toolbar is being served to Community Maps;
- Material Design. (see section 6.3.2 *Look and Feel*)

### 3.1.2 Adding Social Media Connectivity

As social media integration is part of the requirements of WeGovNow, we have implemented on GeoKey core a Social Interaction app where administrators of the projects

<sup>8</sup> <http://www.intenct.nl/projects/django-allauth/>

<sup>9</sup> <https://github.com/ExCiteS/django-allauth-uwum>

<sup>10</sup> <https://github.com/ExCiteS/geokey-wegovnow>

(municipalities) will be able to link their social media accounts for providers such as Facebook or Twitter to their GeoKey account.

- Every time a new contribution is added to the project a new post/tweet will be posted/tweeted to social media.
- User will be able to follow discussions for tweets on social media based on the new contributions added.
- User will be able to pull data from social media and create new contributions based on text search.

When municipalities create a new social interaction they will define the name, description and select the social accounts. They can select multiple social accounts for each social media provider insofar has been authorized previously. Once the social interaction is created they will be able to define the text which will be posted. Tags will include \$category\$ (the category for which the social media post should be made), \$link\$ (the URL that should be included in the post, to allow the user to link back to the map), \$project\$ (the project about which the post is being made) to define the text which will be posted each time a user creates a new contribution for a specific category.

The municipalities will be able to create as many social interactions as they want for each project and link as many social accounts as they want. For example, Torino Municipality links their official Twitter account but they want to post the new contributions in different languages. They will need to create two social interactions (using the \$category\$, \$link\$, \$project\$ pre-defined text above), one where they will define the text to post/tweet in English and the other in Italian.

As well as pushing to social media, we also plan to include data from social media within GeoKey - i.e. to incorporate geo-located posts on our map by pulling data from a social account based on the account name, or on a specific item of text or hashtag. Each of these pulled tweets or posts will be a new contribution and if there is any media attached to the text will be also added to this contribution and added to the map if they are geolocated, or displayed in the project's twitter feed if they are not. We will implement a tool to follow the reactions for each tweet or post.

Currently social interaction models are under development for Twitter and Facebook. We will also investigate the possibility of adding Instagram and Foursquare, as required by the project.

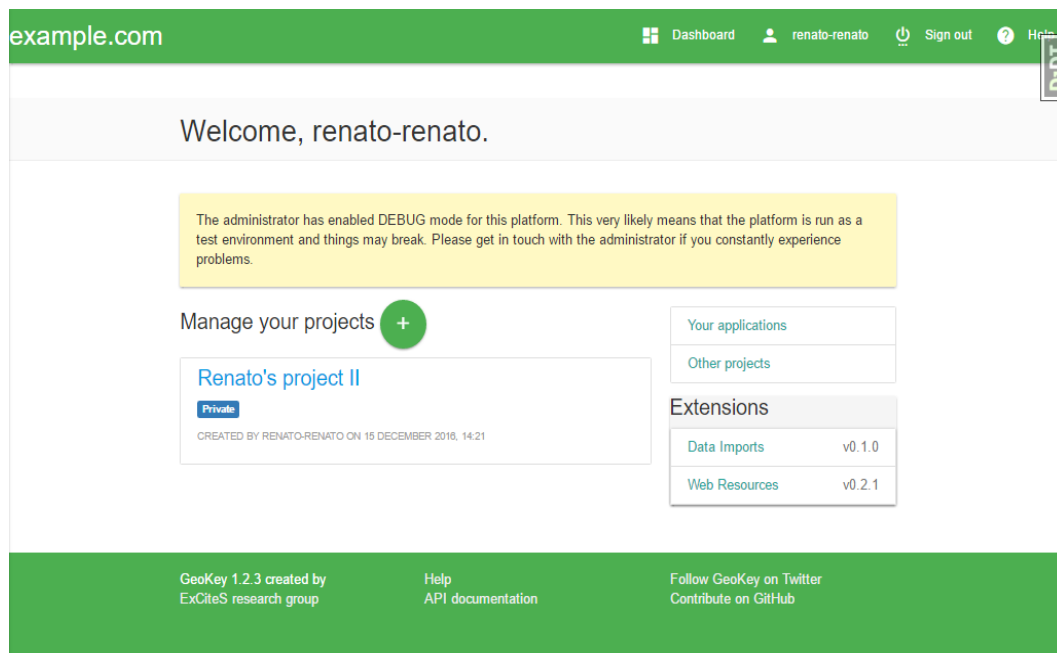
### 3.1.3 Adding Material Design

As part of the requirements of the integration for WeGovNow Project we need to adapt our existing website into Material Design (see section 6.3.2 *Look and Feel*) style because we are currently using bootstrap style.

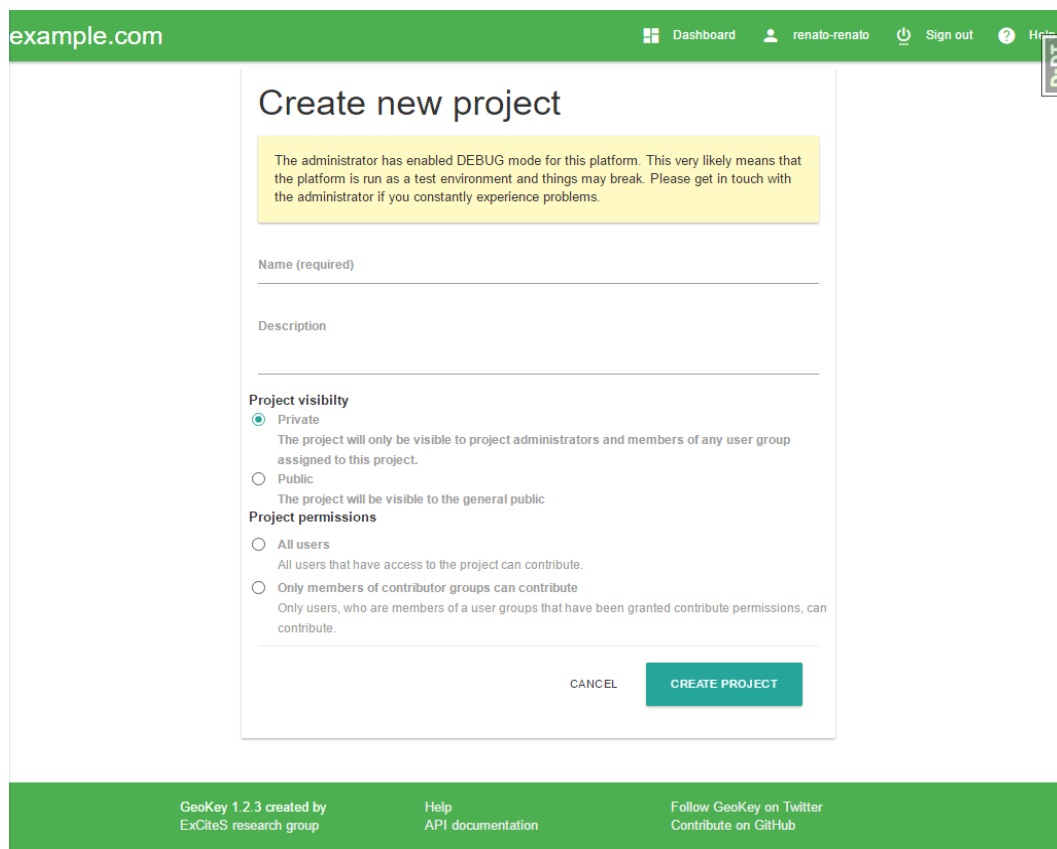
We adapt our website to MD as you can see in Exhibit 17 and Exhibit 18. What we have done so far is to change the existing Bootstrap design to Material Design components. If

required, we will consider a complete redesign of the GeoKey front end to adapt and adjust to the existing Material Design used by websites like Google+, gmail, etc.

*Exhibit 17: GeoKey's projects overview with Material Design*



*Exhibit 18: GeoKey's project creation with Material Design*



## 3.2 Community Maps

Community Maps supports constructing digital representations of physical space through participatory action. Community Maps provides a map-based interface to create, edit and visualise geographic information. Its map interface provides a way in which to add new data as well as editing and deleting existing data. The application further provides a search to find contributions matching a given keyword and filtering according to the status of a contribution. Community Maps is built on top of GeoKey.

Community Maps is a single-page front-end application that connects to GeoKey via the public API. Is it able to retrieve and store public and private information that is visualised onto the map. If private information is to be used, OAuth2 authentication is required to authorise the user.

### 3.2.1 Integrating with UWUM

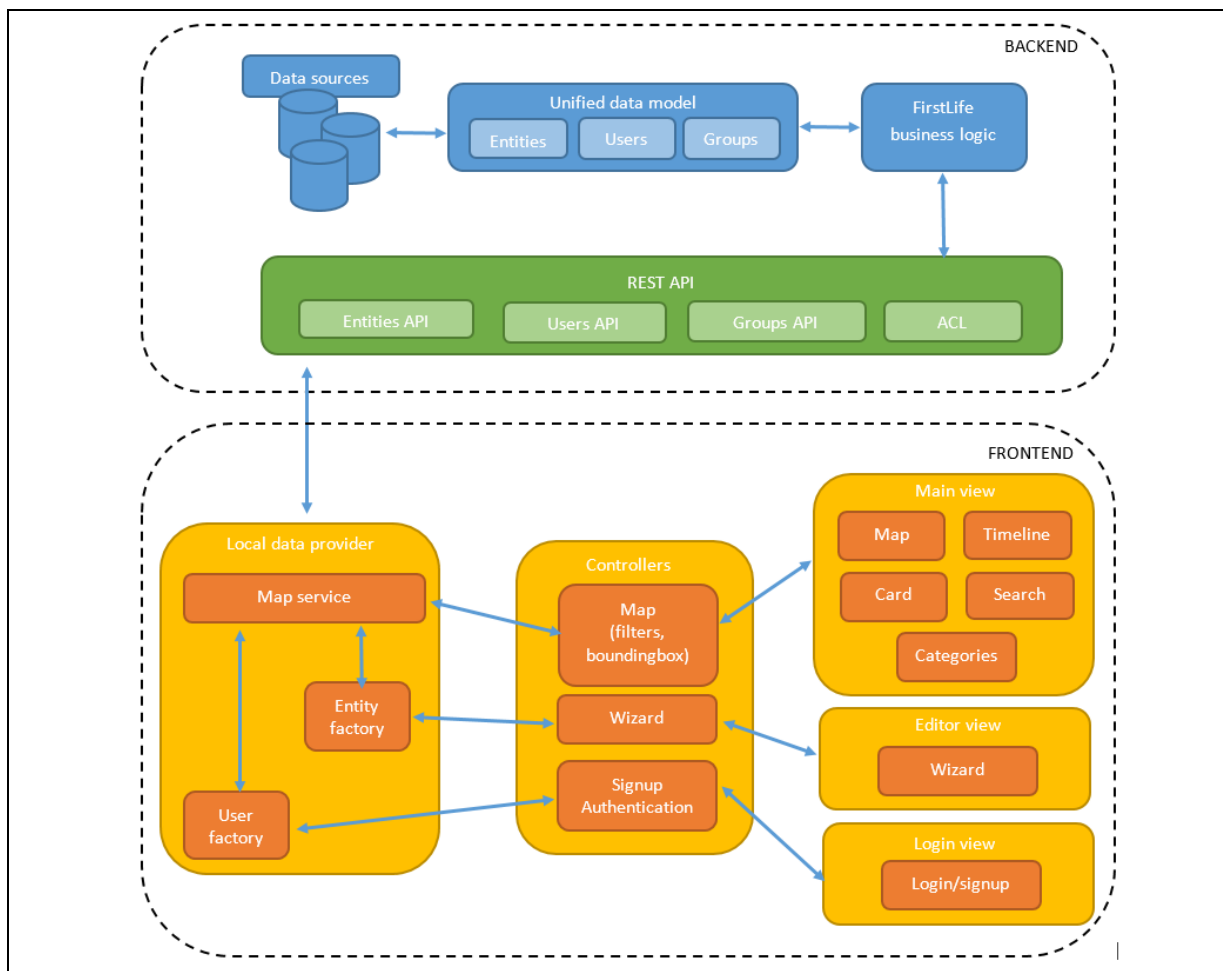
Due to UWUM integration on GeoKey, the authentication architecture of Community Maps was improved. The new architecture allows users to sign in and access user preferences directly on GeoKey. An auto sign in and out feature was also added. The new version of Community Maps also includes the WeGovNow NavigationBar that allows users to navigate from one component to another. The current component is highlighted on the toolbar to inform user that Community Maps is currently in use. Additional Community Maps features (such as a button to change the interface language) were integrated in the WeGovNow NavigationBar.

### 3.2.2 Adding Material Design

Community Maps will adopt the basic Material Design implementation in a future release, which will integrate following the WeGovNow styleguide.

## 3.3 FirstLife

FirstLife is the civic social network for supporting urban networks and institutions cooperation in daily activities developed by UniTO (<http://www.firstlife.org/>). This new type of platform has to address unique challenges in term of enhancing cooperation among heterogeneous local actors, making possible the coexistence of different point of views.

*Exhibit 19: FirstLife client/server architecture and main components*

The FirstLife design process addressed the following guidelines:

- Providing a service layer to FirstLife client and third-part software involved in projects and show cases;
- Collecting external sources of data and providing back data in an open format;
- Supporting the incremental development cycles and projects with flexible structures that can be extended preserving collected data;
- Creating a dynamic map-based view connected to with a multi-dimensional filtering system and relative geographical and temporal queries.

FirstLife implements a client/server architecture, based on REST APIs and latest JavaScript-based technologies such as NodeJS and AngularJS. The figure in Exhibit 19 reports all building blocks of the FirstLife system.

### 3.3.1 FirstLife backend

The backend of FirstLife is developed using LoopBack, a NodeJS based framework that provides a model-oriented approach in which the definition of a high-level data model allows the framework to offers to the user an abstraction layer between the business logic

and database technologies. In this manner, Loopback can supports in a seamless way, different storage technologies, like relational databases, NoSQL, in-memory, etc.

A customizable REST layer allows the user to access, create and modify entities of the model. Currently, FirstLife uses PostgreSQL as storage engine and the pgLatLon for geographical support (see Section 7.1).

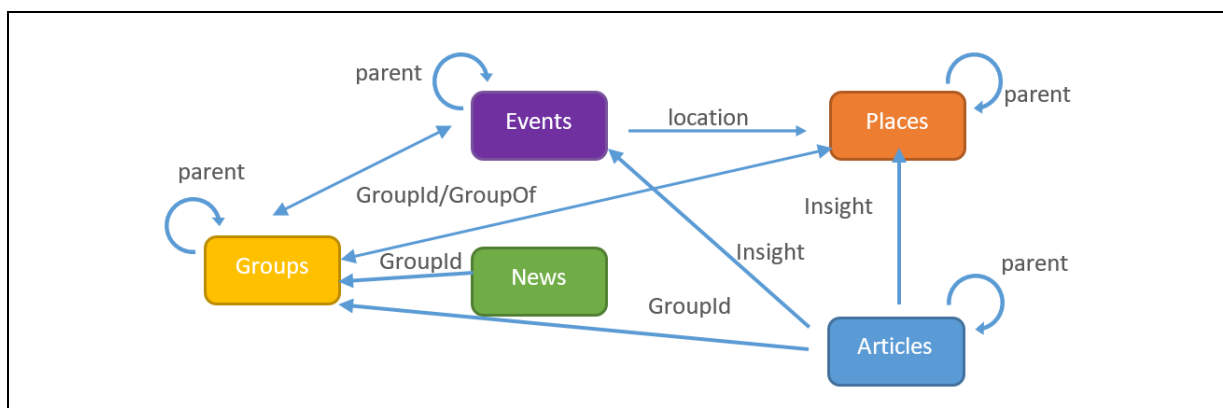
The backend can be invoked by the client or third-part applications via API REST. FirstLife APIs uses a JavaScript Object Notation JSON as message format, in particular an extension of JSON meant for geographical entities: GeoJSON . GeoJSON is a standard format for geographical data, supported by all major GIS (geographical information system) and web GIS software.

The API layer is supported by an access control layer ACL defining users' permissions over contents. The user authentication within the WeGovNow project is provided by the afore mentioned Unified WeGovNow User Management system (UWUM, see Section 2.1). The same module is also responsible for authorization tasks, implementing the well-known OAuth 2.0 Authorization Code flow pattern with the role of "Resource Server". Also in this case, the integration with WeGovNow is ensured by the use of the UWUM framework. All details about FirstLife API can be found in Appendix 3.3.

### 3.3.2 The unified data model

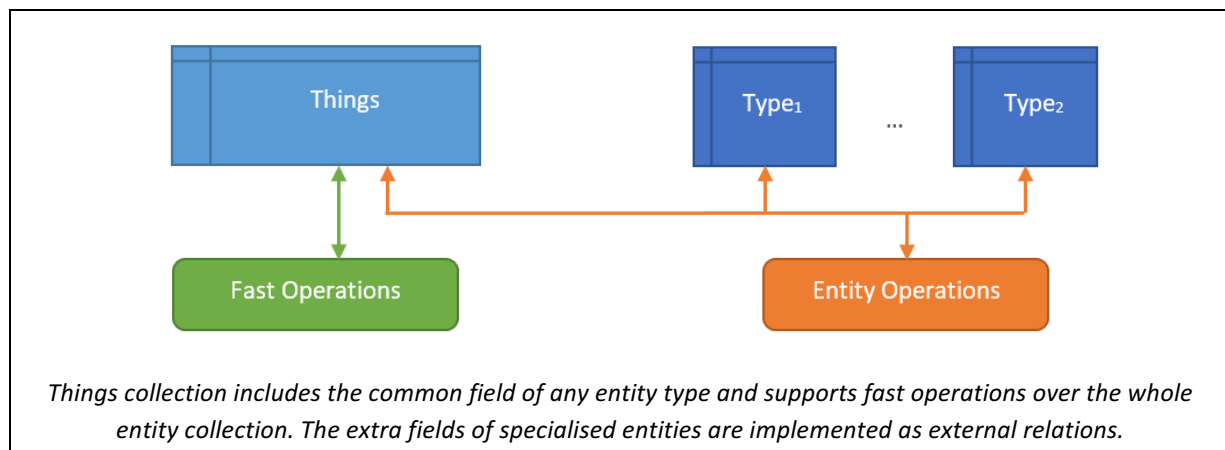
Currently, the FirstLife data model allows the definition of different kind of entities, like places, events, news and articles. All entity types share a common set of properties, like geographical information, ownership, name, categorization, etc. At the same type, each entity type is defined by a set of specific attributes: for examples, events can be described by the starting time and the duration. Moreover, entities can be linked by relations: for example an event is linked to a place by using the relation "location", or a sub-place is linked to its container by a "parent" relationship. Finally, FirstLife allows the definition of Groups. They are meant to collect all other entities, in order to provide a representation of group activities and create a collective responsibility. Groups can also be included in other entities allowing patterns such as group of participants of an event.

*Exhibit 20: Specialised entities in FirstLife and their relations*



In order to keep the data structure flexible and the operations as general as possible we adopted a hierarchical approach based on schema.org structure. We defined a root type “thing” with most of the properties of entities and a set of extensions, one for each entity type. The functionalities were defined over the general entity “thing” making APIs stable during model change. The cost of this approach is related to performances on functionalities based on properties of specialisation entities. In other words, the API works faster when they operate on “things” properties, if the access is required specifically on specialised properties the system needs to rebuild the entities beforehand.

*Exhibit 21: FirstLife database schema implements a hierarchical structure*



### 3.3.3 FirstLife client

The client is a map-based web application relying on well-known technologies like LeafletJs and AngularJS. Users can interact with the map by searching initiatives in a specific area, creating entities and enriching them by adding comments, descriptions and images. As example, in the following figure is depicted a common use of FirstLife in which the user:

1. interacts with the map in order to find the area he is interested in;
2. opens the wall in order to read the titles of the displayed entities;
3. clicks on an entity to;
4. opens a card;
5. adds a description.

*Exhibit 22: Use case for searching and add a new description to an entity*

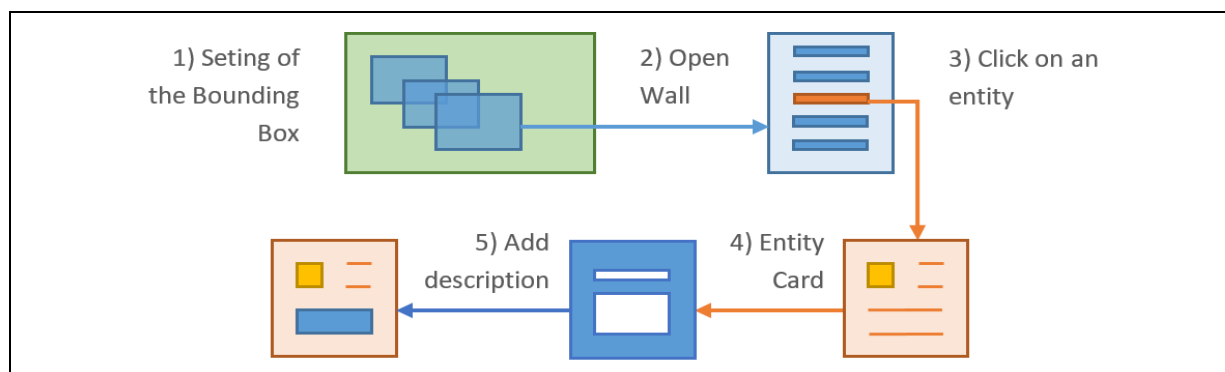
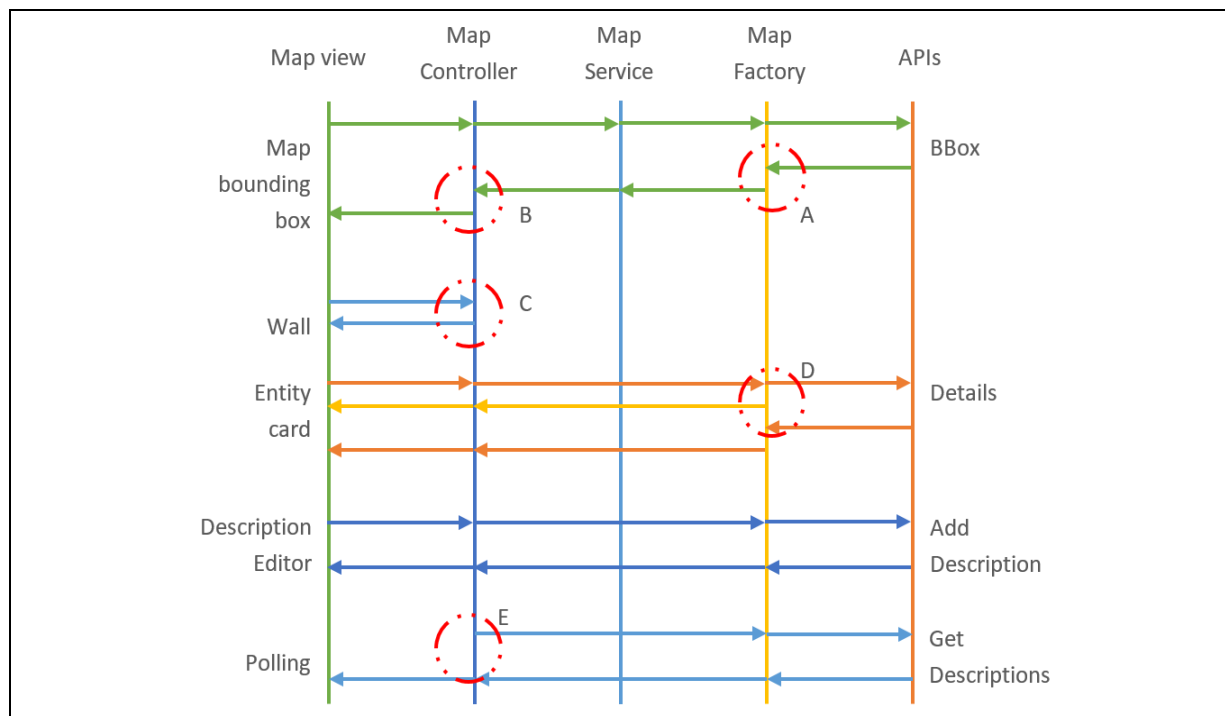


Exhibit 23 depicts the client internal behaviour of a common user interaction. In detail, each interaction with the map generates a bounding box lookup operation that will call the corresponding API endpoint in order to update the client local storage (A). After the update of the local storage, a filter will be applied again on the current set of entities and then sent to the map (B). After, the filtered list will be narrowed using the bounding box, this second list will be visualised in the wall (list of entities) (C).

*Exhibit 23: Information flow of the use case in Exhibit 22, in red the expected delays in client components due to asynchronous calls to the resource server*



A click on an entity within the wall will trigger a “detail” remote call and the opening of the entity card in parallel using the cached incomplete data (D). The retrieved information will complete the entity card. The entity card opening will trigger a temporal request (polling) for description, comments, images (E). Clicking on “add description” a modal with a simple editor will open. After saving the description, the editor will close with a confirmation message and the new description will be included in the entity card.

### 3.3.4 Working with dynamic maps

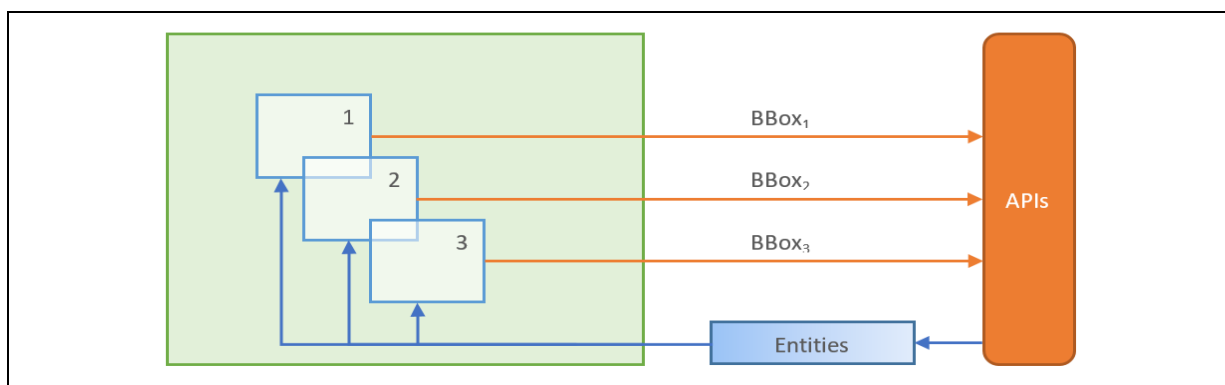
Differently from static maps, a dynamic map requires APIs to populate the web map according to users’ inputs. Together with property filters, such APIs should include temporal and bounding box support in order to filter spatially and temporally data at server side, lowering the memory requirements for the web client.

Static maps preload data on map loading, letting users explore a dataset spatially interacting with a viewer (the map). This approach cannot be used in case of dynamic data.

Moreover, it overburdens a web agent memory and computational resources without any regards to what users actually wish to see. Also, map data should constantly be synched with other users' activities.

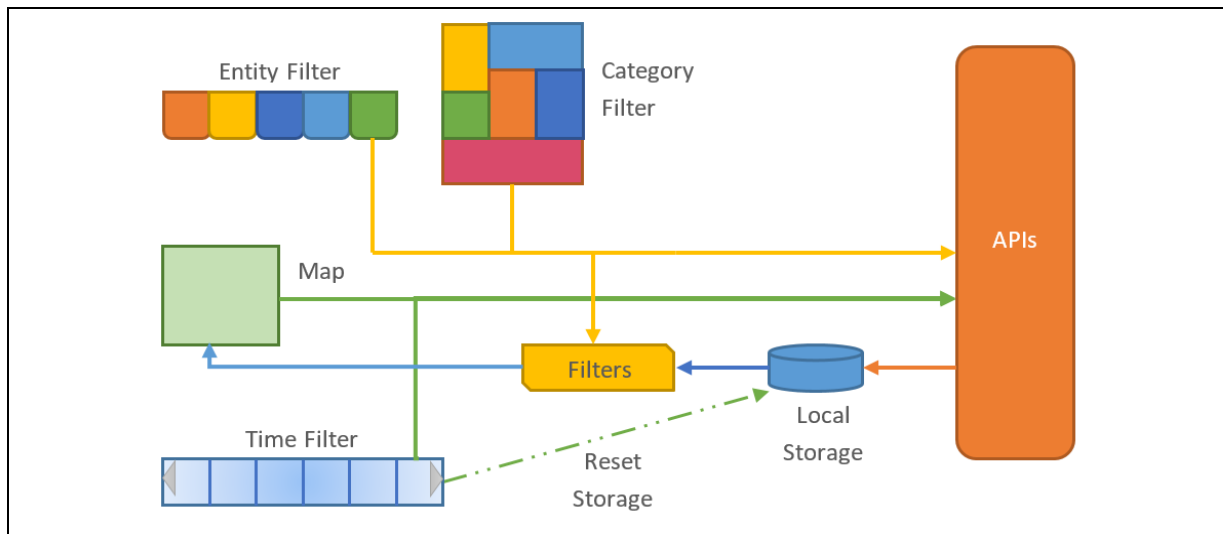
A dynamic web map loads data according to users' inputs. At the end of each interaction (panning, zooming), the map queries a remote storage for the data within the current bounding box. A bounding box is the current view of a map defined by the coordinates of the north/east and the south/west extremes of the box. A bounding box query asks for all contents which geographical features falls within the box.

*Exhibit 24: The change of Bounding Box triggers a query toward FirstLife backend API, the response is consolidated with the local memory of firstlife client before being rendered on the map*

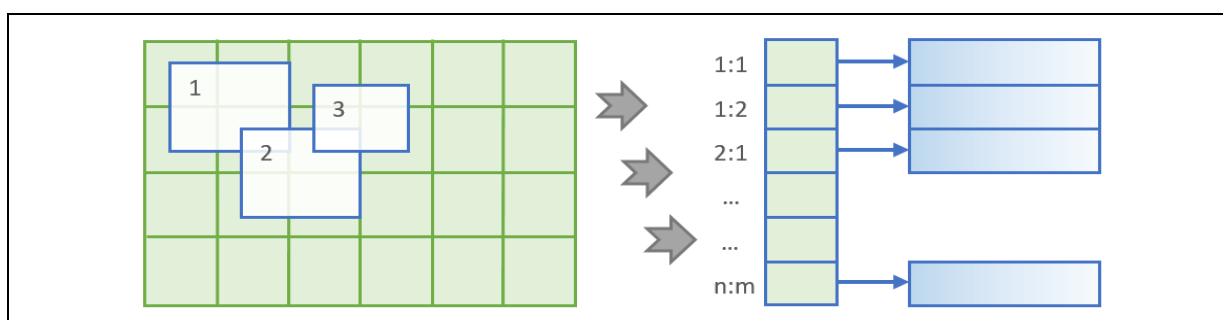


A web client providing a dynamic map has to handle users' inputs, bounding box requests and responses, keep the data updated as they change, manage filtering systems, feed new data into a map and keep eventually delete expired data (Exhibit 25). From the server perspective, geographical types and geographical primitives are complex and still not supported by most of databases. Therefore, the use of geographical primitives needs to be evaluated carefully because of the impact of technological choices and performances.

A web client providing a dynamic map has to handle users' inputs, bounding box requests and responses, keep the data updated as they change, manage filtering systems, feed new data into a map and keep eventually delete expired data. Entity and Category Filters operate on the local storage (entity list) which is rendered on the map. Differently, the input from temporal filters and the map (bounding box) are combined in a bounding box API. The change of the current bounding box will update the local storage, but changes in temporal filters will reset the local data. The results are used to update the local storage, filtered and then sent to the map.

*Exhibit 25: Entity and category filters on FirstLife web client*

It is important to notice that bounding box sessions are not predictable, so caching the results of bounding box queries requires calculating overlaps time wise. For this reason, we are introducing normalisation on bounding box operations using tiles grid. The tile system is a standard division of pyramid images. A tile is a squared section (a standardised bounding box) identified by a triple (x,y,z). The tile system is currently used on dispatching cartography: the cartography is rendered as tiles which are reconstructed on the client side by libraries such as LeafletJS . The web map's main feature is actually the conversion between zoom and bounding box in tile-based queries and the presentation of the map (see Section 2.9).

*Exhibit 26: Tile id is used to index the retrieved data making much more efficient the management at client side*

### 3.4 Improve My City

ImproveMyCity (<http://www.improve-my-city.com/>) is a software platform that promotes direct citizen-government communication and collaboration. ImproveMyCity is an open source, scalable platform that enables residents to directly report, to their public administration, local issues about their neighbourhood such as discarded trash bins, faulty

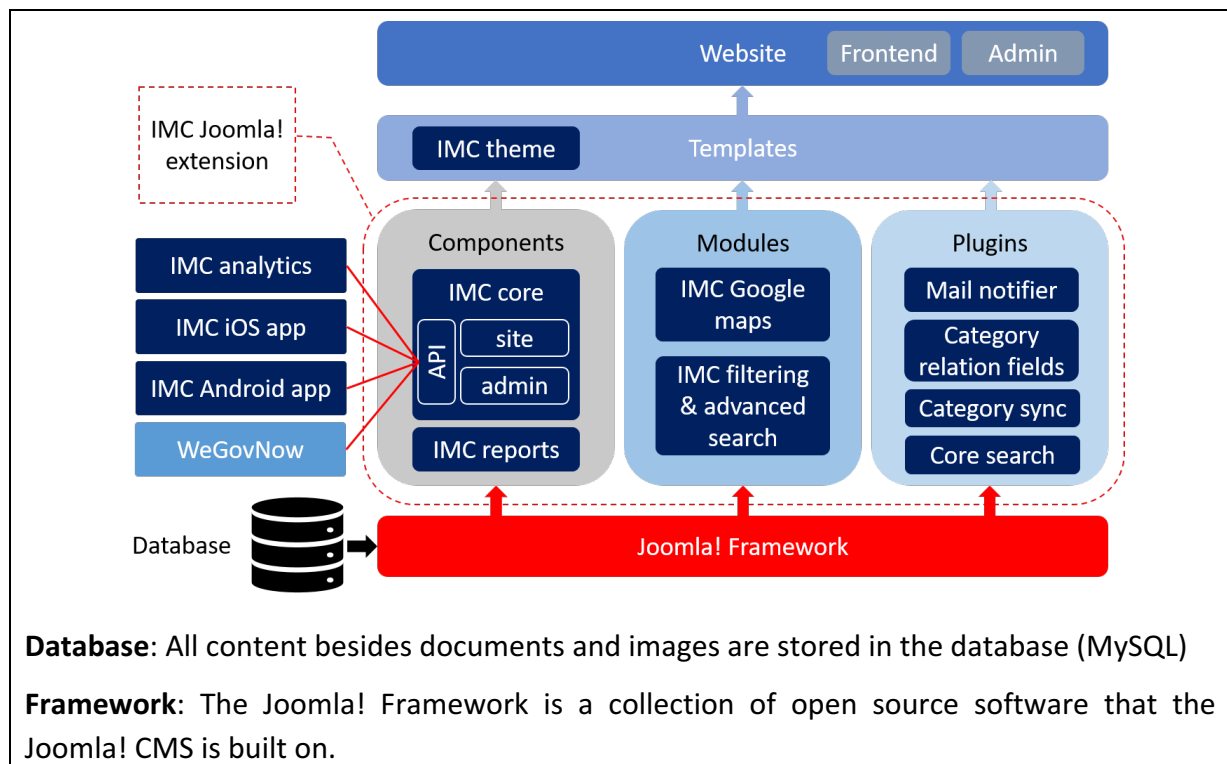
street lights, broken tiles on sidewalks, illegal advertising boards. It also enables submitting ideas and suggestions, and it allows commenting, voting, advanced filtering and transparent timeline of issues' progress. Issues contain geolocation information, but it is feasible to display them as a text only (hybrid approach) list without map.

The reported issues are automatically dispatched to the appropriate department in the public administration to schedule their settlement. The management and routing of incoming issues is performed through a backend administration infrastructure that serves as an integrated management system with easy to use interfaces. ImproveMyCity comes as a Joomla! package extension with an easy to install process, through the Joomla! native installation mechanism of this Content Management System (CMS). There is also a version for WordPress (WP) CMS, available as WP plugin, taking advantage of the friendly user interface of WP. Thus, ImproveMyCity is seamlessly embedded in the most widespread CMS globally. There is also a CMS-free standalone version under development based on a free PHP framework (Laravel).

In WeGovNow, only ImproveMyCity for Joomla! is used, since this version is the most technically and functionally mature one. It provides more features than the other versions, such as complete ACL functionality in the backend administration, hierarchy management and rules and others. Moreover, parts of the external analytics tool might be used during the project if needed.

In the following diagram (Exhibit 27) the architecture of ImproveMyCity as part of the Joomla! framework is depicted.

*Exhibit 27: ImproveMyCity overall architecture as part of the Joomla! framework*



**Components:** Components can be thought of as mini ‘Applications’ and are the place where users mostly interact. Components include the business logic of the applications as well.

**Modules:** Modules are lightweight extensions used in page rendering. They can stand on their own or be used to display data from a Component. Modules are managed from the “Module Manager” (which is itself a Component).

**Plugins:** Plugins are bits of code that execute at specific event triggers. They are a powerful way of extending the Joomla! Framework.

**Templates:** Templates determine the look and feel of the website. They are the ‘icing on the cake’ and they could be different for the frontend/backend but also to different menu items. There is a template available for ImproveMyCity (not part of the ImproveMyCity extension package) that optimises user experience and usability.

**Website:** Website is the interaction point of application logic with users. There is the frontend (what visitors see) and the backend (what administrators see) for the overall management of the system.

ImproveMyCity is a standard Joomla! extension package (depicted in the red dashed box) composed of the items as listed in Exhibit 28.

*Exhibit 28: ImproveMyCity (IMC) package contents*

System name	Item name	Type	Description
com_imc	Improve My City Front-End	Component	The main core component for the front-end that also handles the API controller
com_imc	Improve My City Back-End	Component	The main core component for the back-end (administration)
mod_imcfilters	Improve My City Filters	Module	Includes the buttons for filtering and reordering in any given position (template based)
mod_imcmap	Improve My City Map	Module	Includes the Google Map to display markers with the geolocation of each submitted issue
plg_content_imc	Improve My City Category Fields	Content plugin	Adds relations tab in Joomla! core categories to allow relationship definition between categories and user groups
plg_imc_mail_notifier	Improve My City Mail	IMC plugin	Includes the mechanism to define rules and send notification emails to users

System name	Item name	Type	Description
	Notifier		and administrators based on triggers
plg_search_imc	Improve My City Search	Search plugin	Extends core Joomla! Search to also include ImproveMyCity issues in its results
plg_system_imc	Improve My City Category Timestamp	System plugin	Adds a button in Joomla! Core categories to renew their timestamp (used by the API)
com_imc_reports	*Improve My City Reports	Component	Auxiliary component to produce reports based on date and other filtering. <b>*NOT USED</b> in WeGovNow but it is mentioned for the sake of completeness
imc_analytics	*Analytics	External (acts as a 3rd party client)	Is not part of Joomla installation package but uses ImproveMyCity API to produce analytics, visualisation, and insights. <b>*To be partially used</b> in WeGovNow. Will not be used as is.

The above items could be installed separately but are distributed together as package (besides reports and analytics) with the appropriate installation mechanism packaged in one zip file. All necessary pre- and post- installation actions are taking place during and after installation (e.g. version checking, DB schema updates, automatic updates, etc.) by using Joomla!'s guidelines and best practices as well as by using the default Joomla! installations mechanism for convenience.

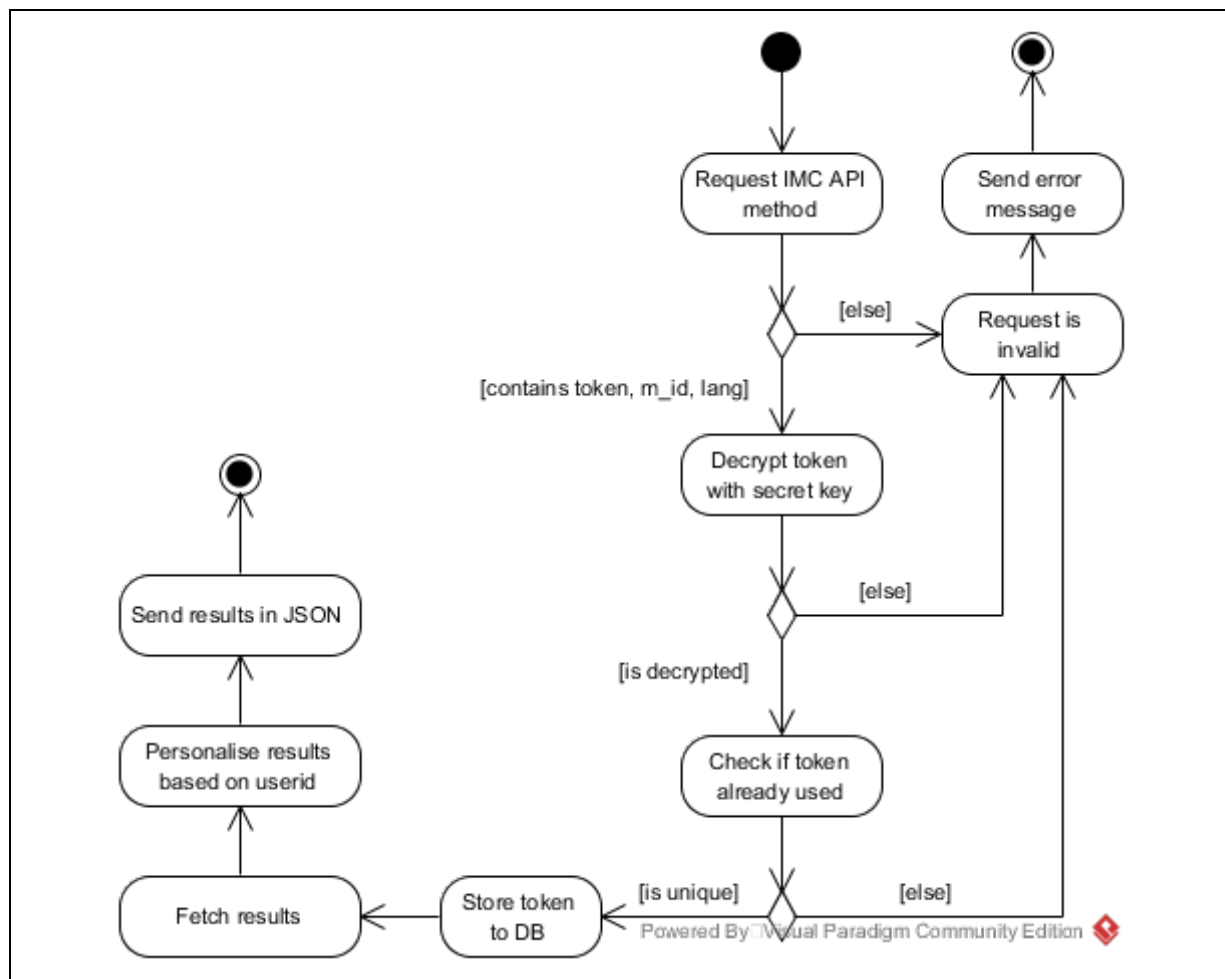
The core part of ImproveMyCity is divided in three conceptual parts; i) the frontend (site), ii) the backend (admin) and iii) the API. The API is handled by a specialised secured Controller (see also section 3.4.1) that allows REST interactions (based on JSON notation) between ImproveMyCity (as a server) and various clients. ImproveMyCity has three clients available, namely i) The analytics tool, ii) the mobile application for iPhones and iii) the mobile application for Android smartphones. The latter applications will not be part of WeGovNow (as they are proprietary closed-source software). For the needs of WeGovNow and its requirements in terms of integration, the API has been significantly updated as will be explained in section 3.4.2.

### 3.4.1 ImproveMyCity API calls and security

Exposing the complete functionality of ImproveMyCity through its API means that extra actions should be taken in terms of security. There is a complete custom mechanism to ensure security between ImproveMyCity (acting as a server) and its clients (e.g. analytics, mobile applications, and other 3rd party platforms like WeGovNow). The dataflow of the

ImproveMyCity token-based security mechanism is depicted in the activity diagram shown in Exhibit 29 below.

Exhibit 29: ImproveMyCity API request data flow



First of all, the secret key exchange is occurring once and in offline mode. Generally, it is a process that is handled in-house and the secret key is actually hard-coded at client side. Then, every single call to ImproveMyCity API has to include; the encrypted token, the modality ID and the language code:

*Token is the m-crypted "json\_encode(array)" of username, password, timestamp and random text in the following form:*  
*{'u':'username','p':'password','t':'1439592509', 'r':'VxxXZR0dR9'}*  
*all casted to strings including the UNIX timestamp time()*

*where m\_id is the modality ID according to the REST/API key definition in the administrator side*

*where lang (l) is the 2-letter language code used for for the*

*responses translation (en, el, de, es, etc). Moreover, every token is allowed to be used only once to avoid MITM<sup>11</sup> attacks.*

ImproveMyCity uses the Advanced Encryption Standard (AES) and the exact class that it uses can be found in Github<sup>12</sup>. It is based on Rijndael-128 algorithm. ImproveMyCity API calls are stateless, meaning that no sessions are stored and that is the reason that encrypted user credentials should be contained in every single API request.

For ImproveMyCity installations that are strictly stand-alone, the API Controller (as part of the MVC<sup>13</sup> pattern) that handles all REST communications can be completely removed without conflicts if desired.

Every client that connects to ImproveMyCity could have its own secret key set in the administrator side (with easy GUI) for extra security and easy logging and monitoring (applied quota restrictions is also feasible). This security mechanism can function without SSL being enabled but this is not suggested.

### 3.4.2 Extending ImproveMyCity for WeGovNow

In order to take full advantage of ImproveMyCity features and functionalities within the WeGovNow ecosystem, ImproveMyCity is extended in various aspects. A list of all updates, modifications and improvements that took place already or will be completed in the next phase are briefly listed below;

1. UWUM integration in terms of i) authentication, ii) validation, iii) styling
  - a. Automatic local user creation and interlinking with UWUM member\_id.
  - b. Automatic login/logout of users based on CORS requests.
2. Integration of the Navigation Bar as Joomla! Module. It can be freely used by any Joomla! installation and thus help to spread WeGovNow community by allowing any 3rd party application that is based on Joomla! and wants to join WeGovNow, to be able to do it fast and reliable.
3. API with UWUM validation and automatic creation of local users.
4. Introduce GeoJSON from/to ImproveMyCity API requests and responses.
5. Creation of the UWUM slogin plugin for Joomla! It can be used by the Joomla! open source community and by any existing Joomla installation. It also includes a graphical user interface (GUI) to setup easily.
6. Creation of the UWUM login button (at the same manner as “Login with Facebook” button) to be used by 3rd party Joomla-based applications that wants to join WeGovNow.
7. Creation of an OpenStreetMap based Joomla! module to replace Google Maps module and allow easier integration of WeGovNow AreaViewer.

---

<sup>11</sup> MITM stands for “Man In The Middle” and refers to common hacking attack technique. ImproveMyCity takes security measures to avoid such attacks by not allowing security tokens to be used more than once.

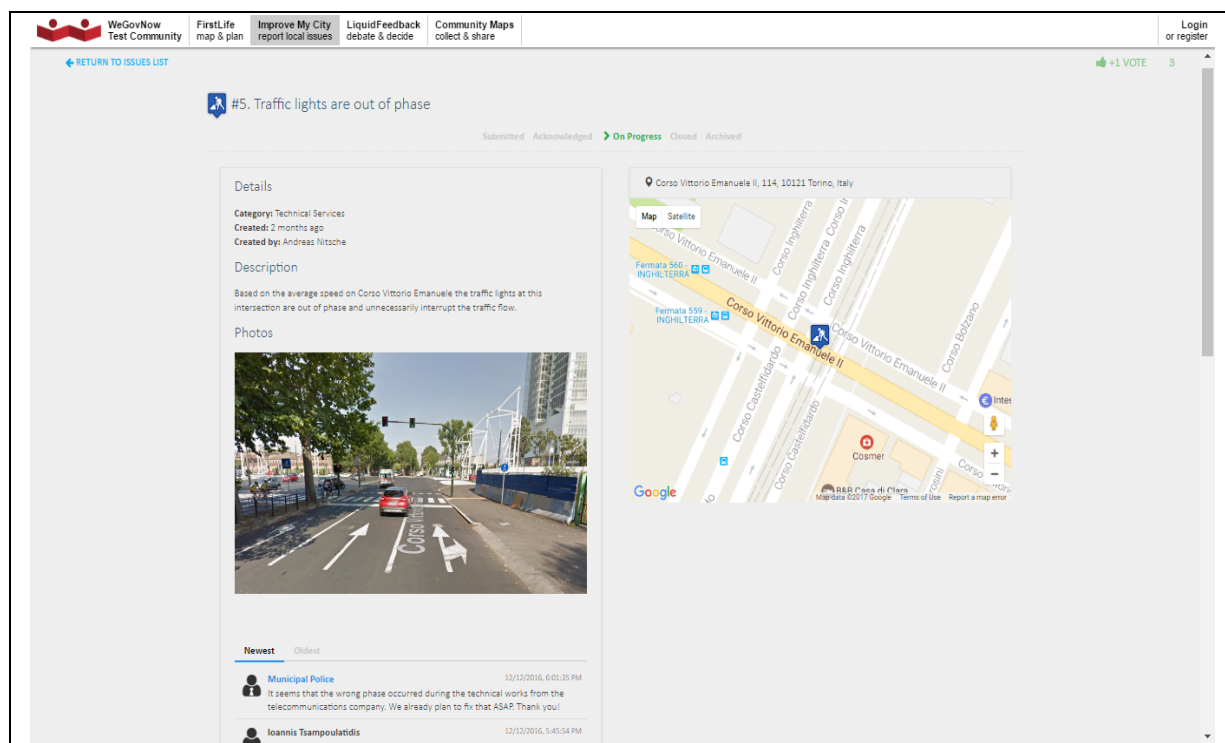
<sup>12</sup> <https://github.com/itsam/imc/blob/master/site/helpers/MCrypt.php>

<sup>13</sup> MVC stands for Model-View-Controller. Is a broadly used software pattern and is also the approach that is followed by ImproveMyCity (and Trusted Marketplace)

8. Creation of a Joomla! template to match the look 'n feel of WeGovNow using Material Design.
9. Accessibility improvements based on Funka's report for ImproveMyCity.
10. Applying the Material Design guidelines on ImproveMyCity component and modules.
11. Create Dispatchers to notify OnToMap Logger with user actions as Joomla! Plugin that acts as Observer to various triggers (such as new comment, status change, etc).
12. Deploy automated scripts to allow ImproveMyCity installation by Command Line Interface (CLI). It is a future proof mechanism that is planned to be used by WeGovNow Service Discovery and WeGovNow dynamic installation. Also a method to comply with WeGovNow licensing (see also D1.2).
13. Creation of ImproveMyCity Sandbox (available at the very early stages of the project) to allow pilot sites to experiment with the features and functionalities of ImproveMyCity and to allow technical partners to monitor the progress in terms of new features integration.
14. Interactive API documentation based on Swagger API framework to allow technical partners to test the new ImproveMyCity API requests approach (based on UWUM).

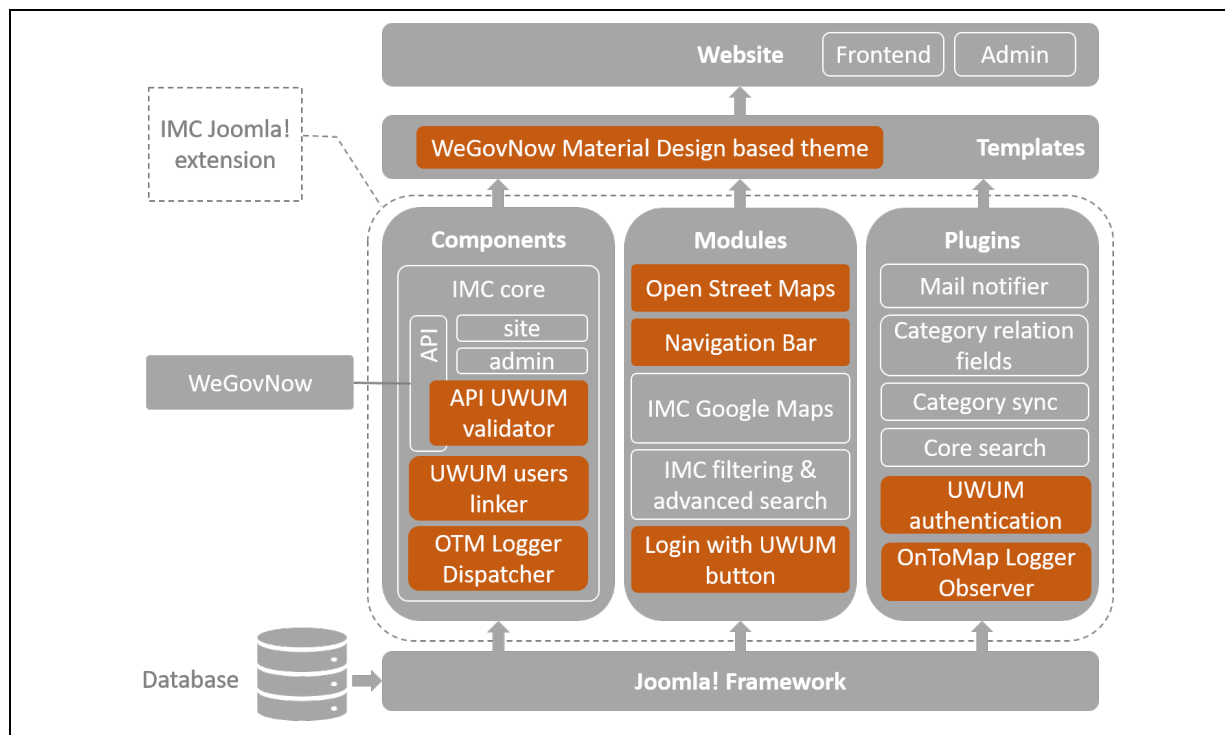
ImproveMyCity for WeGovNow (Exhibit 30) is available online at <https://wegovnow.infalia.com>

*Exhibit 30: Screenshot of ImproveMyCity with Navigation Bar and Material Design applied*



The new approach and all the extensions that took place in ImproveMyCity towards WeGovNow integration are depicted in Exhibit 31 (in comparison with Exhibit 27).

Exhibit 31: Extension of ImproveMyCity architecture towards WeGovNow integration



Having in mind the extendibility and high modularity of ImproveMyCity, most of the new features towards the WeGovNow integration are implemented as modules and plugins. The main advantage of this, besides clean code and easier maintenance and testing, is the ability to make these features publicly available to the open source community of Joomla! and thus allow future applications to bypass the complexities involving for example UWUM integration. Modules and plugins are already available in Github<sup>14</sup>. The repository will be constantly updated as more features and functionalities are mature enough to be exposed and promoted accordingly. The highlighted boxes in Exhibit 31 above show the new functionalities that are introduced in ImproveMyCity. These are briefly described in the following paragraphs.

**Templates:** The WeGovNow compatible template includes a specialised position just for the Navigation Bar and it is applied with Material Design guidelines. It is planned to further update the template taking into account the advice of project partner Funka on accessibility-related issues.

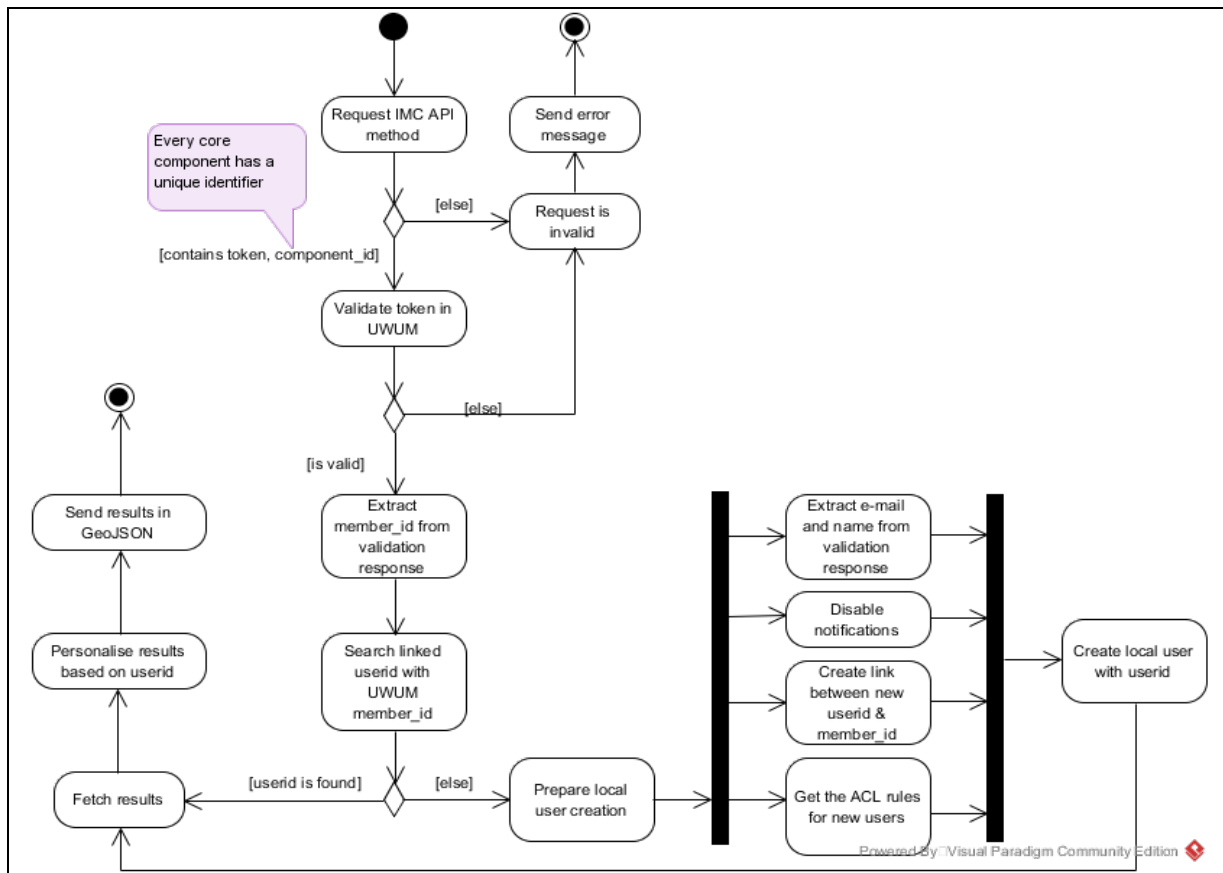
**Modules:** i) The “**Navigation Bar**” module is already available and includes the latest updates in terms of adaptive design. It incorporates the Cross-Origin Resource Sharing (CORS) mechanism that checks the user state based on their OAuth2.0 session at UWUM side. ii) The “**Login with UWUM**” module is available for Joomla-based 3rd party applications that are willing to join WeGovNow. It is not directly used in ImproveMyCity, (besides debugging), since its functionality is included in the “Navigation Bar” module. iii)

<sup>14</sup> <https://github.com/infalia>

The **“Open Street Maps”** module is planned to be implemented for easier integration of the WeGovNow AreaViewer component and also for being consistent with the rest of the WeGovNow core components that do not use Google Maps (which is the current mapping system of ImproveMyCity).

**Component’s new features & Plugins:** i) The **“UWUM authenticator”** handles all necessary actions concerning OAuth2.0 authentication, handling the certificate, keeps track of the session and refreshes when the security token is expired. This plugin is linked with the **“UWUM users linker”** feature in core components and allows the interlinkage between local ImproveMyCity users and UWUM users (userid  $\leftrightarrow$  member\_id). This is an advanced feature that is necessary to keep the business logic, Access Control List (ACL) and personalised filtering (e.g. “show only my issues”) of ImproveMyCity, intact. ii) The **“OnToMap Logger Observer”** plugin that is planned to be released prior to first prototype in M15, handles all triggered actions from **“OnToMap Logger Dispatcher”** in ImproveMyCity core. The role of this plugin is to collect user activity in real time and send it over to OnToMap Logger to be further processed. Lastly, the **“API UWUM validator”**, is the most extended update towards WeGovNow integration. The approach of calling ImproveMyCity API methods as presented in comparison to Exhibit 29 has changed significantly. Exhibit 32, depicts the new approach on requesting the ImproveMyCity API.

*Exhibit 32: New approach on ImproveMyCity API methods invocation for WeGovNow*



As shown in the above activity diagram, the former custom based security token has been abandoned. Instead, the standard OAuth2.0 approach through UWUM has been applied. There is a lot of extra functionalities that are hidden under the hood when following the new approach though. Besides having to change all API methods in the core component, the automatic creation of users when those are registered in another WeGovNow core component (e.g. FirstLife, LiquidFeedback, etc) is a quite complicated procedure and has to take place transparently without being noticed by the method caller. The new approach on API methods is in progress.

The complete list of ImproveMyCity API as an interactive documentation is available at <http://wegovnow.improve-my-city.com/api/> along with guidelines on how to invoke the methods online.

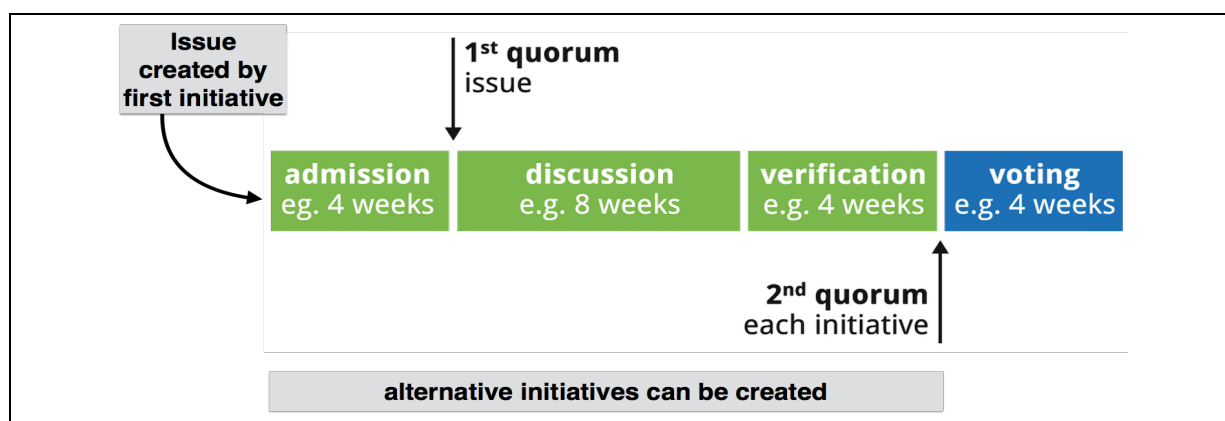
### 3.5 LiquidFeedback

LiquidFeedback is a web application for the democratic self-organisation of large scale groups. It combines concepts of a collectively moderated, self-organized discussion process (quantified, constructive feedback) and liquid democracy (delegated or proxy voting).

LiquidFeedback covers the process from the introduction of the first draft of a proposal to the final decision. This way it allows to participate not only in voting but also in developing ideas. Discussing an issue before voting increases the awareness of pros and cons, chances and risks, and allows people to consider and suggest alternative trade-offs which can become part of the final voting procedure.

Extra effort has been made to ensure minorities can express their view and stay visible. On the other hand, LiquidFeedback can also handle the challenge of noisy minorities.

*Exhibit 33: The structured proposition development and decision making process in LiquidFeedback*



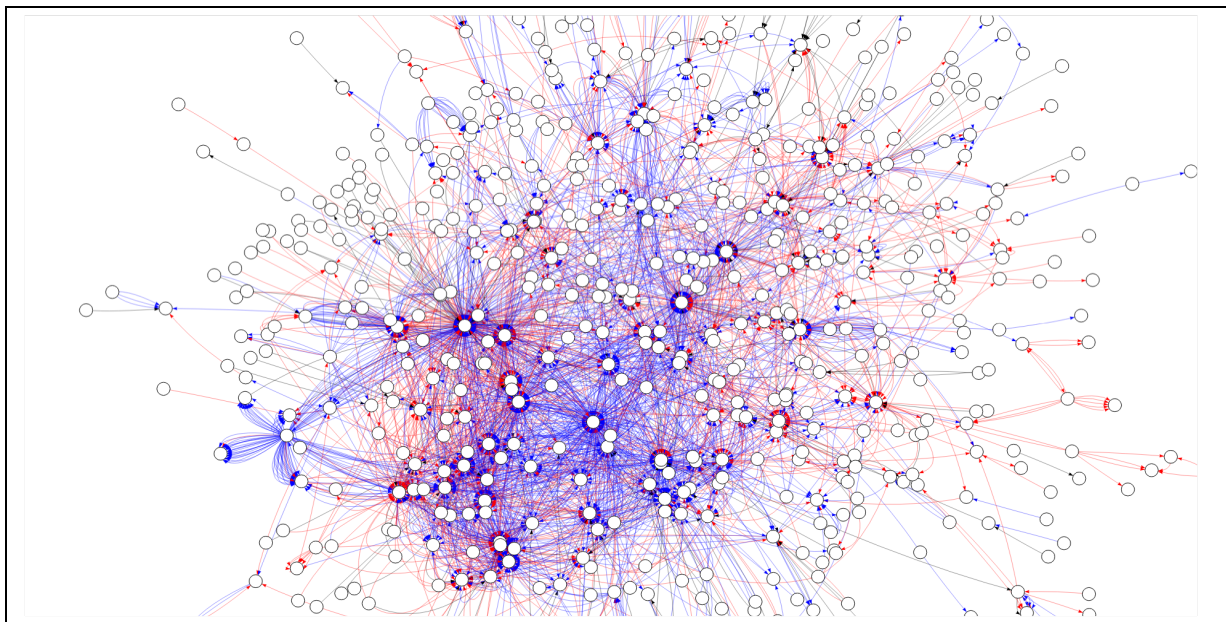
One distinct feature is the support of a dynamic division of labour based on individual choice: Voters can delegate their vote to a trustee (technically a transitive proxy). The votes can be further delegated to the proxy's proxy thus building a network of trust. All

delegations can be done, altered and revoked by topic. A dynamic scheme of representation takes place.

In LiquidFeedback delegation can be given

- for all issues in all subject areas within an organisational unit (e. g. counties, districts, boroughs or parts thereof),
- for all issues in a particular subject area (e.g. “traffic”) of an organisational unit,
- for a single issue, which is a group of already existent competing proposals to be voted upon together.

*Exhibit 34: Dynamic division of labour: Delegations for units, areas and issues*



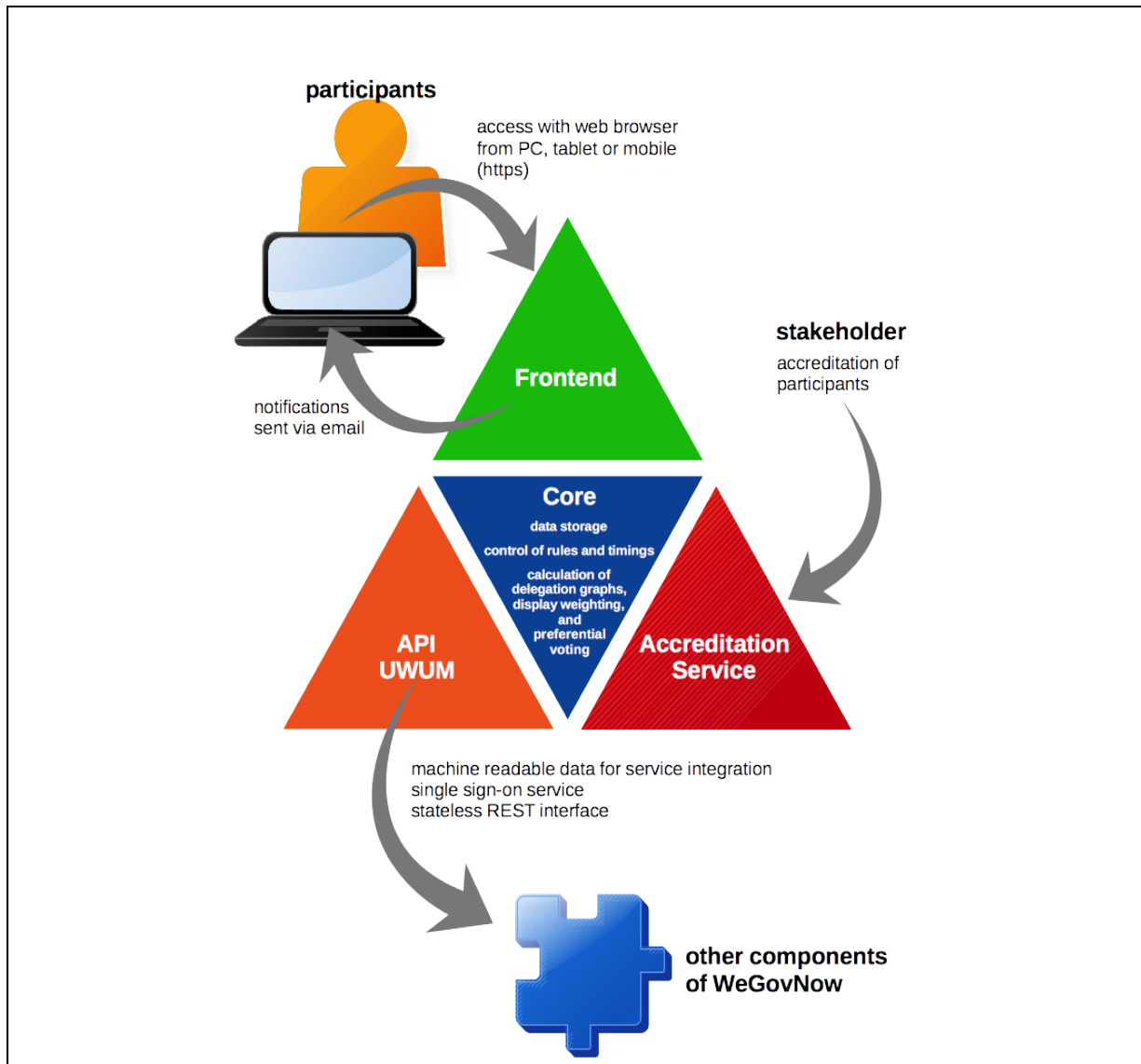
Delegations are not only applied to the final vote on a given issue but also applied during its discussion, where it is possible to rate other people’s proposals. Whenever a more fine-grained delegation exists (e.g. a delegation for a particular issue), a more general delegation (e.g. a delegation for the respective organisational unit) is overruled for the affected subject area or issues. The same holds when you make use of your own vote: If you enter a discussion by supporting or opposing proposals or make your own proposals, then you can’t delegate your vote during discussion. If you participate in a final voting, you can’t delegate in that final voting. Any form of direct participation will suspend existing delegations. (A proxy can not vote in the presence of the principal.) The LiquidFeedback process and the dynamic division of labour are explained in a dedicated publication, “The Principles of LiquidFeedback”<sup>15</sup>.

<sup>15</sup> Behrens, Kistner, Nitsche, Swierczek: “The Principles of LiquidFeedback”. ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V.

### 3.5.1 LiquidFeedback Core extensions

LiquidFeedback consists of a database backend (which is called “LiquidFeedback Core”) and a user interface (called “LiquidFeedback Frontend”). While data processing such as vote counting is performed by the backend, the web interface for the user is provided by the LiquidFeedback Frontend. Interfaces to other applications, such as the other WeGovNow components are realized through a newly developed REST API.

*Exhibit 35: Architecture of LiquidFeedback as WeGovNow component*



As a prerequisite for implementing a REST API for LiquidFeedback, the backend (“LiquidFeedback Core”) had to be modified in regard to the vote counting rules (allowing for an integration with other components) as well as geo-tagging features (enabling geo-spatial queries).

LiquidFeedback version 3.x relies on explicit member registration per subject area in order to calculate a reference population that is then used in the 4-phase decision making

process. Registration in a subject area, however, is infeasible when integrating LiquidFeedback with map-based frontends. Some users might not even be aware of the available subject areas in LiquidFeedback, hence the registration counts would be arbitrary, which, in turn, would influence LiquidFeedback's decision making process in a non-predictable way. Previous work has been published on a possible modification of the LiquidFeedback decision making process<sup>16</sup>. It was decided by the LiquidFeedback developers to generally adopt such a modification while some simplifications were proposed. A working implementation, however, was still outstanding, and had thus to be created as part of WP3. For details refer to section "Further details of implementation" below.

LiquidFeedback 3.x has no means of storing geographical information. In order to allow for LiquidFeedback initiatives and suggestions to be displayed on a map, the necessary data fields needed to be added to the data structures of the LiquidFeedback backend ("LiquidFeedback Core").

### 3.5.2 Planned REST API including geo-spatial queries

The planned REST API for LiquidFeedback needs to include search functions (e.g. boxes parameterized with latitude/longitude ranges, or radial searches with a centre and a radius). Considering a possibly huge number of datasets, spatial indexing is a requirement for offering such API calls. LiquidFeedback up to now has not used a database with geospatial capabilities.

Considering accuracy, performance, dependency and licensing issues, LiquidFeedback decided to implement pgLatLon, a spatial database extension for the PostgreSQL object-relational database management system providing geographic data types and spatial indexing for the WGS-84 spheroid.

While many other spatial databases still use imprecise bounding boxes for many operations, pgLatLon aims to support more precise calculations for all implemented geographic operators. Efficient indexing of geographic objects is provided using space-filling fractal curves. Optimizations on bit level (including logarithmic compression) allow for a highly memory-efficient non-overlapping index suitable for huge datasets.

pgLatLon is a lightweight solution as it only depends on PostgreSQL itself (and a C compiler for building). Unlike competing spatial extensions for PostgreSQL, pgLatLon is available under the permissive MIT/X11 license to avoid problems with viral licenses like the GPLv2/v3. pgLatLon will also be used by partner UniTo in FirstLife.

Democratic requirements demand further functions of a geo-spatial database: when working with user-generated content, users may be tempted to create intentionally oversized objects in order to optimize search results in an unfair manner. The newly

---

<sup>16</sup> "A Finite Discourse Space for an Infinite Number Of Participants", Liquid Democracy Journal, Issue #4, ISSN 2198-9532

created “pgLatLon” software includes a prototype for a “fair distance” function, which aims to handle such illegit user input by returning an adjusted distance (i.e. distance increased by a penalty) if a geographic object consists of more than one point.

### 3.5.3 Further details on implementation

Details on the core extension can be found in the “Work report on extending the LiquidFeedback Core” in Appendix 3.5. Details on pgLatLon can be found in section 7.1 and the “Work report on pgLatLon, an alternative to PostGIS” in Appendix 3.7.1. Background on the “fair distance” function can be found in the reference (section 4 of the README file) of pgLatLon version 0.10, lines 472 through 520<sup>17</sup>.

### 3.5.4 UWUM and integration framework

LiquidFeedback’s API provides the Unified WeGovNow User Management (UWUM) and Integration Framework described in sections 2.1 through 2.6 and in the “Work report on Unified WeGovNow User Management (UWUM) development” in Appendix 2.1.2 of this document.

## 3.6 OnToMap

The following description presents OnToMap in synthesis. More information can be found in the live document at the following URL:

[https://drive.google.com/open?id=1XGfjf7lhQ9NtWbT-eLMDi1\\_WeUI\\_L6Ui11gBTwWqE1g](https://drive.google.com/open?id=1XGfjf7lhQ9NtWbT-eLMDi1_WeUI_L6Ui11gBTwWqE1g).

OnToMap<sup>18</sup> is a web-based application that supports the consultation of spatial data, the creation of virtual, interactive maps reflecting individual information needs, and the creation of custom project maps. An important feature of OnToMap concerns knowledge representation, which supports data integration and multi-faceted information retrieval by exploiting a semantic representation of geographical concepts and relations. Within the WeGovNow platform, the OnToMap backend is used in two ways:

- As an **Open Data Container** (original functions provided by the OnToMap application); see below.
- As a centralised **activity logger and data integrator** (new functions, developed in the WeGovNow project), in order to support centralised user activity logging and data retrieval across applications (see this section and section 2.2).

The map management and user interface functions offered by the complete OnToMap application are not included in the WeGovNow platform.

---

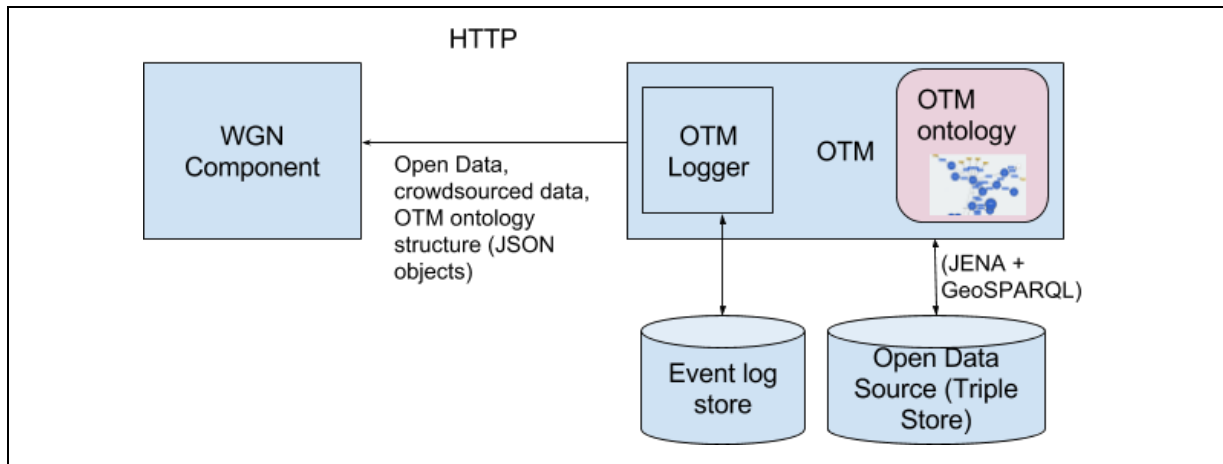
<sup>17</sup> <http://www.public-software-group.org/mercurial/pgLatLon/file/v0.10/README.mkd>

<sup>18</sup> <https://ontomap.ontomap.eu>

### 3.6.1 OnToMap backend architecture

OnToMap can integrate a set of Open Data sources by exploiting a semantic knowledge representation layer that makes it possible to reconcile data representational and conceptual heterogeneity.

*Exhibit 36: Architecture of the OnToMap backend*



The OnToMap ontological knowledge representation layer makes it possible to:

- Integrate heterogeneous data, originated from different sources, and manage it as Linked Data. As reported in <http://linkeddata.org/>, "Linked Data" is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods."
- Describe semantic relations among information items to express spatial relations, different levels of abstraction in the description of entities and thematic relations. This approach enables a semantic and geospatial exploration of the information space.

In line with consolidated approaches adopted in GIS for data integration<sup>19</sup>, and recognizing the growing importance of the OWL Web Ontology Language for semantic knowledge representation, the OnToMap Ontology is defined using the OWL Web Ontology Language<sup>20</sup>. Moreover, Linked Data is represented as RDF (Resource Description Framework<sup>21</sup>) triples. This makes it possible to exploit standard tools for browsing the ontology and for retrieving geographical information items.

As shown in Exhibit 36, which presents the OnToMap backend architecture, OnToMap stores Linked Data in the Open Data Store: this is a triple store, i.e., a specialized tool for

<sup>19</sup> For instance, see (i) Fonseca, F. T. "Using ontologies for geographic information integration" Transactions in GIS, 2002, and (ii) K. Janowicz, S. Scheider, T. Pehle, and G. Hart. 2012: Geospatial Semantics and Linked Spatiotemporal Data – Past, Present, and Future. *Semantic Web - On linked spatiotemporal data and geo-ontologies*

<sup>20</sup> <https://www.w3.org/OWL/>

<sup>21</sup> <https://www.w3.org/RDF/>

managing and querying RDF datasets. The current implementation of OnToMap exploits the Parliament triple store<sup>22</sup>, which supports GeoSPARQL queries<sup>23</sup>.

OnToMap exploits the Apache Jena ontology API<sup>24</sup> to perform GeoSPARQL queries on ontology concepts. The result of a query is a set of RDF triples representing the geographical information items satisfying the query. This set of triples is translated to an external GeoJSON format to make data directly usable by WeGovNow applications for map-based presentation.

The OnToMap Ontology has been developed by analyzing and reconciling heterogeneous terminologies, such as those derived from a number of Open Data sources, and those adopted by the WeGovNow applications; see section 7.3.

### 3.6.2 Data retrieval and integration

The provision of a single access point to the information available in the WeGovNow platform is carried out by merging the results of two processes:

- *Retrieval of the Open Data integrated in the platform:* The Logger queries the Open Data Store to retrieve the instances of the requested concepts and translates Linked Data to the GeoJSON format used for data exchange within the WeGovNow platform.
- *Retrieval of data crowdsourced by WeGovNow applications:* The storage of detailed user activity events, whose arguments are expressed in the terminology of the OnToMap Ontology, is the key element for integrating crowdsourced data across the WeGovNow platform. By translating the arguments of each user activity to the OnToMap Ontology format, we get as a by-product the storage of a unified description of the datum which the actions are about; e.g., the updated version of an existing information item, see the example in Table 2.6.1. Notice that, as each logged event reports complete information about the activity object and the timestamp of the activity, the Logger can retrieve the latest version by checking the timing of events (and, in general, retrieve a previous version by checking timestamps of events). Obviously, the integrated data only concern information that has been shared by WeGovNow applications by pushing the related user events to OnToMap Logger.

However, the integration of different data sources raises the “multiple identity” issue in the description of geographical entities. In fact, the sources might provide separate descriptions of the same entity, and associate different deep links (external URLs) to it. For the management of multiple identities, OnToMap provides a **WeGovNow Entity Registry** that stores a tuple for each set of URIs referring to the same geographical entity, that are available in the WeGovNow platform. Data consolidation is thus performed as follows:

- Recognition (and storage into the WeGovNow Entity Registry) of the identity of multiple geographical objects belonging to the Open Data, by analyzing and comparing

---

<sup>22</sup> <http://parliament.semwebcentral.org/>

<sup>23</sup> <http://www.opengeospatial.org/standards/geosparql>

<sup>24</sup> <https://jena.apache.org/documentation/ontology/>

the data stored in the used Open Data Sources. This task, performed using heuristics on the data sets (based on the geometry, category, name of the objects), makes it possible to create tuples of URIs denoting the same entities:  $\langle \langle \text{URI1}, \text{URI2} \rangle, \langle \text{URI3}, \text{URI4}, \text{URI5} \rangle, \dots \rangle$ .

- Recognition of the identity between the geographical objects managed by WeGovNow applications and the URIs belonging to the WeGovNow Entity Registry, and possible revision of the registry. This is done by exploiting the WeGovNow InputMap provided by FirstLife; see Section 2.5.

The Logger uses the WeGovNow Entity Registry to retrieve information about entities acquired in the whole WeGovNow platform, by performing integrated queries that refer to the complete set of equivalent URIs.

### 3.6.3 OnToMap data retrieval API

The interaction between OnToMap and the other WeGovNow applications, regarding data retrieval and semantic data representation, is managed through the API specified at <https://ontomap.eu/>, which returns results as a JSON object.

The API is available only to authorized WeGovNow applications, and requires certification: every request must include a X.509 client certificate signed by LiquidFeedback.

The API can be used to navigate the common data structure defined by the OnToMap Ontology and to retrieve the Open Data provided by the OnToMap Open Data Store, as well as the data crowdsourced by WeGovNow applications, in the unified terminology and format provided by the OnToMap Ontology. Specifically:

- The API supports the exploration of the concepts and relations stored in the OnToMap Ontology. It makes it possible to retrieve the root concept of the ontology, the list of concepts, the concepts related to a specific concept  $C$ , and so forth.
- The API can also be used to retrieve information about the instances of a concept  $C$ , which can be expressed either in the terminology of the WeGovNow application (in which case OnToMap translates  $C$  to the corresponding concept in the OnToMap Ontology), or directly in the OnToMap Ontology format. Depending on the filters used in the query, the retrieved instances can belong to the Open Data and/or to the data crowdsourced by the WeGovNow applications. The API supports filtering instances by geographical area and/or temporal validity, and/or data source (Open Data belonging to a certain source, crowdsourced data). The geographical information items returned by the API are represented as GeoJSON “Feature”<sup>25</sup>, with “hasType” and “external\_URL”, as described in Section 2.2.

The tables below provide two examples of API invocation and show subsets of the results returned by OnToMap. It should be noticed that the invocation path might change in future development steps and has to be considered as an example.

---

<sup>25</sup> See <http://geojson.org/geojson-spec.html#feature-objects>.

*Exhibit 37: Sample result of the invocation of OnToMap Data Retrieval API to get the concepts of the OnToMap Ontology*

### Example 1: Listing all the concepts of the OnToMap Ontology

API invocation: <https://api.ontomap.eu/api/v1/concepts?relations=true&prettyprint=true>

Returned result (we report a small subset of the retrieved results for readability purposes):

```
{
  "concepts" : [ {
    "uri" : "http://ontomap.eu/ontology/PedestrianPath",
    "label" : {
      "en" : "Pedestrian Path"
    },
    "description" : {
      "en" : "A type of thoroughfare that is intended for use only by pedestrians and
not other forms of traffic such as motorized vehicles, cycles, and horses."
    }
  }, {
    "uri" : "http://ontomap.eu/ontology/TrainStation",
    "label" : {
      "en" : "Train Station"
    },
    "description" : {
      "en" : "A place along a route or line at which a train stops to pick up or let
off passengers or goods, especially with ancillary buildings and services."
    }
  }, {
    "uri" : "http://ontomap.eu/ontology/ProvincialRoad",
    "label" : {
      "en" : "Provincial Road"
    },
    "description" : {
      "en" : "A road maintained by a province."
    }
  }, {
    "uri" : "http://ontomap.eu/ontology/Logistics",
    "label" : { },
    "description" : { }
  }
  .... other concepts etc. ....
]
}
```

*Exhibit 38: Sample result of the invocation of OnToMap Data Retrieval API to get the instances of a concept of the OnToMap Ontology*

### Example 2: Listing all instances of an OnToMap Ontology concept (UrbanPark)

API invocation:

<https://api.ontomap.eu/api/v1/instances/UrbanPark?descriptions=true&page=0&prettyprint=true&geometries=true>

Returned result -: we report only two urban parks for readability purposes:

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "id" : "http://ontomap.eu/ontology/UrbanPark/1752",
    "application" : "ontomap.eu",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 7.70454410999532, 45.116544469625 ], [ 7.70458372336213,
45.1166866489942 ], [ 7.70484456454166, 45.1167625573604 ], [ 7.70523767471388,
45.1166818256333 ], [ 7.70525338232746, 45.1165876942459 ], [ 7.70552345587221,
45.116611244572 ], [ 7.70552869809785, 45.116581265747 ], [ 7.70521582757928,
45.1165545025648 ], [ 7.70521889868876, 45.116538451576 ], [ 7.70520882592985,
45.1165374911834 ], [ 7.70514639189535, 45.1164604316532 ], [ 7.70454410999532,
45.116544469625 ] ] ]
    },
    "properties" : {
      "external_url" : "https://ontomap.eu/api/v1/instances/UrbanPark/1752",
      "hasType" : "UrbanPark",
      "hasCharacterization" : "VERDE VARIO",
      "hasScope" : "CENTRO DI INCONTRI",
      "hasName" : "EDIFICIO COMUNALE",
      "hasManagement" : "DI QUARTIERE",
      "hasID" : "1752",
      "label" : {
        "it" : "Parco Urbano - EDIFICIO COMUNALE"
      }
    }
  },
  "provenance" : "http://ontomap.eu/ontology/Provenance/Torino2014"
} ]
}
```

### 3.7 Trusted Marketplace

Trusted Marketplace is the only core component that wasn't available prior to WeGovNow project. Instead, it is being designed and implemented from scratch according to the requirements and needs of the pilot use cases. Besides the major role of Trusted Marketplace that is the **match-making** of users-to-events and users-to-users and the handling of **personalised notifications** (based on these match-making suggestions), Trusted Marketplace also incorporates features that are not available by the other core components. These features include:

- Enhanced user profile management.
- Mechanism to handle demands and offers with a built-in reputation system.
- Graphical representation of the personalised timeline by aggregating OnToMap Logger user actions/activities in a friendly interface.
- Mechanism to define, globally in the overall WeGovNow platform, personalised accessibility preferences to promote the 'Design For All' principles and guidelines.
- A friendly dashboard for easy overview of all the above.

In the following diagram (Exhibit 39) is depicted the high level architecture of the Trusted Marketplace and its interaction to other components of WeGovNow.

The Trusted Marketplace engine receives input from three different sources.

1. Indirectly from any other core component (including the "Offers & Demands Organiser, part of Trusted Marketplace) through the OnToMap Logger. All user events (actions and activities) are requested and being pulled by the Logger into the "Event Decomposer" that decomposes the raw consolidated information into data about i) activities, ii) location, iii) timestamp and iv) description as free text. The spatial data are forwarded to the "Geo Relevance Scorer", the temporal data are forwarded to the "Temporal Relevance Scorer", the interest/activity data are forwarded to the "Activity Scorer", and data about user reputation are forwarded to the "Trust Ranker".
2. Directly from the users through the "Dashboard" and their personal profile, preferences and interests that are explicitly declared by them.
3. By social networks, on user's consent, through the "Social Media Linker & Collector".

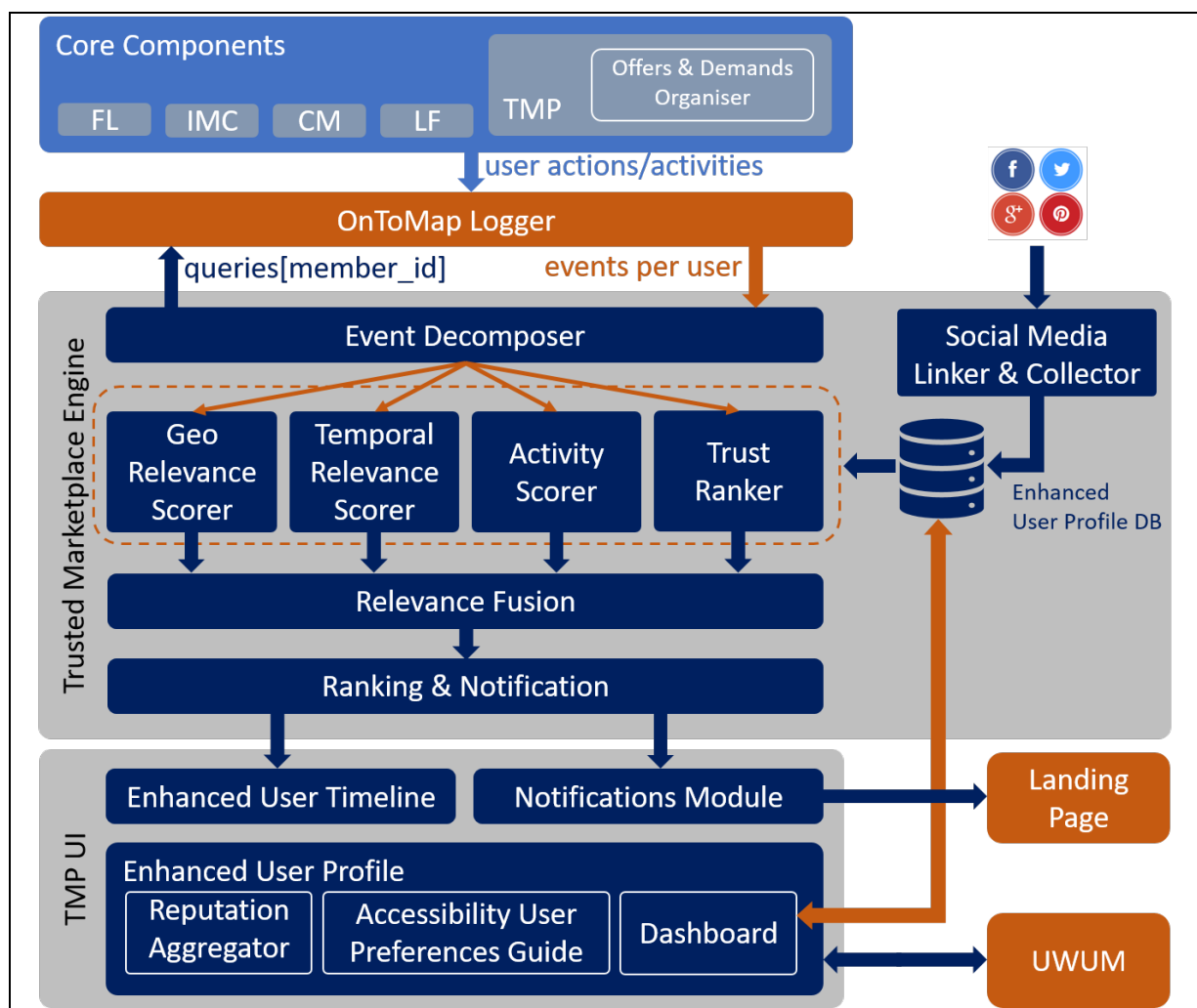
The "Relevance Fusion" merges the various scores that are calculated by applying several rules such as (but not limited to): i) certain radius containing actions of all the places that the specified user has recorded activity (geolocated), ii) actions within available time-slots (e.g. every weekend), iii) reputation threshold, iv) relevance matching threshold, v) notification frequency threshold (e.g. up to 3 notifications per day) and others. Lastly, the "Ranking & Notification" feeds the "Notification" mechanism that informs users with personalised suggestions in the Landing Page, and also helps to enhance the personalised

timeline in the user interface (UI) of the Trusted Marketplace. It also helps to prioritise (rank) the most relative offers/demands according to user interests.

Trusted Marketplace UI also includes i) the “Reputation Aggregator” which is actually part of the “Offers & Demands Organiser”, ii) the “Accessibility User Preferences Guide” and iii) the Dashboard. The latter three modules are composing the “Enhanced User Profile” that interacts with the UWUM to get and set global variables (e.g. user preferences) as key-value pairs.

In order to make the multiple roles of the Trusted Marketplace more clear, Appendix 3.7 contains a set of mockups that illustrate how Trusted Marketplace is graphically envisioned after discussing with partners from the pilot sites and having their scenarios unfolded. These mockups will be finalized prior to the first prototype and the major implementation phase is expected to be finalised by the second prototype phase.

*Exhibit 39: High level architecture of Trusted Marketplace (TMP)*



## 4 Component integration

The WeGovNow platform is the result of the integration of several stand alone components in an environment providing support to sharing resources and information among components. WeGovNow features are the union of the features of its component plus new features available thanks to the coexistence and synergy of the components.

The integration within WeGovNow enables the creation of new features based on the communication and/or the orchestration of components' functionalities. In this regards, the core features of WeGovNow are mostly designed to facilitate the integration as follows:

- The unified user management is to share users and exchange user's information and preferences.
- The application discovery service is to retrieve at run time the current WeGovNow setup.
- The style service is to personalise the component at run time.
- The navigation bar in data and html format
- The schema translation service and the unified logger are to retrieve the activities from all other components.
- The InputMap is to easily interconnect application entries.
- The AreaViewer is to provide general summaries.
- The common set of open data provided by municipalities as linked open data

In addition to the core features, the consortium agreed (see Appendix 5) on four main patterns in interconnecting components:

1. Deep linking, interconnection based on URLs;
2. Data embedding: including data from other applications;
3. View embedding: including a pre-rendered views generated by other applications;
4. Trigger actions: using functionalities provided by other applications via API.

In order to support the engagement activities providing a scope of features broader than the sum of WeGovNow components, and in order to find as early as possible architectural constraints and limits, the technical teams produced a joint assessment of how to combine the component features.

In general, the approach we follow in WeGovNow is based on standards and design patterns. But in case of building cross component features a joint analysis of the mechanics used in each component and of the possible interactions was required. In the following subsections, the results of bilateral assessments concerning the combination of each of the WeGovNow component are presented.

## 4.1 Community Maps/GeoKey ⇄ WeGovNow Components via OnToMap and AreaViewer

The GeoKey/Community Maps Integration with all WeGovNow components will happen in a systematic manner. This approach does not only enable integration with existing components. It also ensures that the integration approach pursued is scalable and future proof for any new components potentially to be added to the WeGovNow family. Integration will occur on three levels, as follows:

- Firstly, via the common, shared menu and common UWUM login and user management environment (already in progress);
- Secondly, via the aggregator/overview map (see AreaViewer). Specifically, the aggregator map will query data from GeoKey (we will extend our API to support e.g. queries that include spatial bounding boxes, object validity dates and so forth) and then serve this aggregated data out to the other WeGovNow components as GeoJSON (either raw or as a 'summary' depending on the need of the requesting component) or as a ready-made embeddable map.

Similarly, GeoKey can query data from the Aggregator API to get summary data from the other components, which will provide links back to the disaggregated data within the individual components so that this can be seen in its real context. Clicking on the data in the AreaViewer will offer the user the option to go to the source component for the data to see the detail.

- Thirdly, we will integrate with OnToMap via the logger. Our integration with OnToMap is based on a bilateral agreement after providing OnToMap with a full list of what we can deliver.

Taking this approach, we develop a scalable solution for WeGovNow as all any new components have to do is connect to UWUM, connect to the aggregator/overview (we probably need a better name, and a standard minimum set of API calls) and the OnToMap logger. That way, there will be very little (or even no!) requirement for additional programming when a new component is to be added.

## 4.2 FirstLife ⇄ LiquidFeedback and FirstLife ⇄ ImproveMyCity

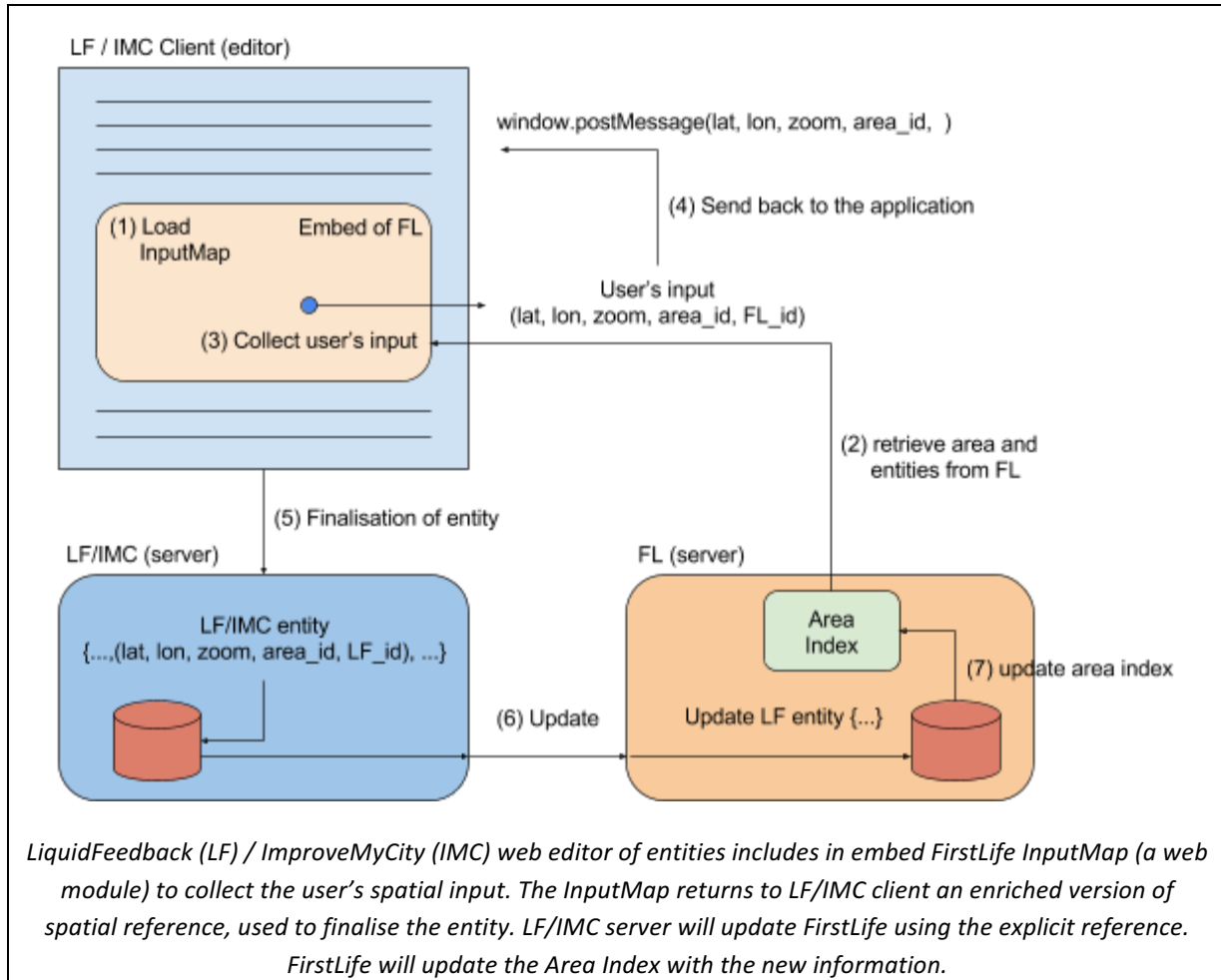
The bilateral integration between FirstLife and LiquidFeedback, and between FirstLife and ImproveMyCity builds on four functionalities:

- Association of initiatives (LiquidFeedback) or issues (ImproveMyCity) with a FirstLife geo feature;
- Sharing initiative/issues data with FirstLife;
- Display of initiatives/issues in FirstLife;
- Displaying FirstLife map in LiquidFeedback and ImproveMyCity.

During the creation and update of initiatives/issues, LiquidFeedback and ImproveMyCity will embed a geo feature editor provided by FirstLife. This way a user of LiquidFeedback or ImproveMyCity can add a geo feature to the initiative/issue. The geo feature is stored in

FirstLife while a reference to the FirstLife geo feature is sent back to LiquidFeedback or ImproveMyCity respectively and stored there.

*Exhibit 40: Bilateral integration between FirstLife and LiquidFeedback, and between FirstLife and ImproveMyCity*



FirstLife receives a continuous stream of initiative/issue updates from LiquidFeedback and ImproveMyCity. FirstLife can store initiative/issue data associated to geo features referenced.

FirstLife shows initiative/issue geo features along with other geo features. FirstLife can use and/or show additional data from LiquidFeedback and ImproveMyCity. FirstLife can provide a popup for each geo feature showing a teaser (using pre-rendered partial views via API). The popup teaser view in FirstLife includes a "more" link to LiquidFeedback's initiative or ImproveMyCity's issue view (deep linking).

Several views in LiquidFeedback and ImproveMyCity will include a FirstLife map using an appropriate viewport to show the geographical context.

Obviously, this integration approach can be adopted by a large number of applications which do not have native mapping features. This way these applications can easily add mapping functionality without having to deal with the complexity of mapping.

### 4.3 FirstLife $\rightleftharpoons$ OnToMap

The interaction between OnToMap and FirstLife is managed by the OnToMap Logger, according to the schema depicted in Exhibit 5 **Error! Reference source not found.** on page 19. The goal is that of enabling FirstLife to:

- Push to OnToMap Logger information about user activities on FL, which FirstLife agrees to share in the WeGovNow platform; e.g., creating, updating, removing, commenting geographical objects. The user activities concern geographical objects described according to FirstLife domain conceptualization; i.e., they refer to the FirstLife data categories.
- Retrieve a history of actions performed by users that one or more WeGovNow applications previously pushed to OnToMap Logger. E.g., FirstLife manipulations of geographical objects, ImproveMyCity issues, LF initiatives, etc..
- Retrieve the descriptions of geographical objects available to OnToMap Logger: this consists of the information reported in the history of events stored by the Logger + the Open Data stored by OnToMap.

The interaction between FirstLife and OnToMap Logger is carried out by invoking the API described in Section 2.2 of this deliverable. FirstLife agrees to provide user activity events and geographical data descriptions in the required format.

All geographical object descriptions returned by OnToMap Logger are represented in GeoJSON format and refer to the concepts of the OnToMap Ontology. This requires data integration from WeGovNow applications and OnToMap Ontology. FirstLife is eligible for this integration because it has a stable domain conceptualization which can be mapped to the OnToMap Ontology.

The OnToMap Logger APIs require FL's X.509 client certificate signed by LiquidFeedback. Moreover, the Logger assumes that the user IDs occurring in the pushed events are valid: i.e., users have been authenticated by FirstLife before pushing the activity information to OnToMap Logger.

### 4.4 ImproveMyCity $\rightleftharpoons$ LiquidFeedback

ImproveMyCity and LiquidFeedback will take advantage of their integration with FirstLife and OnToMap:

- Issues from ImproveMyCity will automatically show up in LiquidFeedback
  - Initiatives and Issues from LiquidFeedback will automatically show up in ImproveMyCity
- User of either application will be aware of the existence of issues and/or initiatives in the other application.

This approach is compliant with the consortium decision to define “FirstLife as a map component to provide a consolidated map for all non mapping applications preferably based on the logger or ontology provided by OnToMap”.

If no appropriate mapping application is present ImproveMyCity and LiquidFeedback would have to fall back to a plain OpenStreetMap approach with coordinates rather than geographical objects. The absence of a mapping application in a future WeGovNow installation is very unlikely. Therefore ImproveMyCity and LiquidFeedback assign a low priority to this alternative scenario.

#### 4.5 LiquidFeedback $\rightleftharpoons$ OnToMap and ImproveMyCity $\rightleftharpoons$ OnToMap

The interaction between OnToMap and LiquidFeedback as well as OnToMap and ImproveMyCity is channelled through the OnToMap Logger, according to the schema depicted in Exhibit 5 **Error! Reference source not found.** earlier in this document. The integration builds on the following concepts:

1. LiquidFeedback and ImproveMyCity actively push information about user activities which shall be shared with other WeGovNow components to the OnToMap Logger.. The user activities are specific for the application and can include deep links to geographical objects held by a WeGovNow application e.g., a FirstLife geographical entity that the initiative/issue is about:
  - a. LiquidFeedback: *creating an initiative, updating an initiative, creating a suggestion, voting on an issue, delegating a user, user registration/profile events, etc..*
  - b. ImproveMyCity: *submit a new issue, edit an issue, vote an issue, comment on an issue, update the status of an issue, change the category of an issue, upload files to an issue, get the timeline of an issue.*
2. LiquidFeedback and ImproveMyCity can retrieve a history of actions performed by users that one or more WeGovNow applications previously pushed to OnToMap Logger, e.g. FirstLife manipulations of geographical objects, ImproveMyCity issues, LiquidFeedback initiatives.
3. LiquidFeedback and ImproveMyCity can retrieve the descriptions of geographical objects available to the OnToMap Logger which consist of the information reported in the history of events stored by the Logger and the Open Data stored by OnToMap.

The interaction between the application and OnToMap Logger is carried out by invoking the API described in Section 2.2. LiquidFeedback and ImproveMyCity will provide user activity events and geographical data descriptions in the required format.

All geographical object descriptions returned by OnToMap Logger are represented in the GeoJSON format and refer to the concepts of the OnToMap Ontology.

The OnToMap Logger’s API requires authorization through an X.509 UWUM client certificate. Moreover, the Logger assumes that the user IDs occurring in the pushed events

are valid: i.e., users have been authenticated by the application before pushing the activity information to OntoMap Logger.

## 4.6 OnToMap $\rightleftharpoons$ TrustedMarketplace

The interaction between the two components is similar to LiquidFeedback  $\rightleftharpoons$  OnToMap and ImproveMyCity  $\rightleftharpoons$  OnToMap (see section 4.5). More specifically, Trusted Marketplace implements a distributed event handling system and follows the Dispatcher/Observer pattern. The events of Trusted Marketplace include: *submit a new offer, submit a new demand, submit score, submit a new comment, update profile, profile enriched (automatically), link new social account, remove social account, new match occurred (automatically), new match (manually), new notification sent, new interest added*.

Concerning the pull of data from OnToMap to TrustedMarketplace; this deals mainly with the dynamic graphical representation of the personalised timeline. Trusted Marketplace requests by UWUM “member\_id” + “range of timestamps” and receives raw information of all actions occurred to/from the specified user from every connected core component. Trusted Marketplace by comparing their interests, neighborhood, etc.), also by merging with data collected by social networks.

## 5 Technology and user driven scenarios

In its final form, of the WeGovNow platform will need to be adopted by stakeholders in local communities and individual end users. When defining the platform architecture, it was therefore necessary to anticipate the utilisation of the WeGovNow platform by the pilot municipalities and other stakeholders within their day-to-day activities. Before engaging users, a general assessment of WeGovNow platform architecture was performed with a view to identifying new possibilities offered by the interaction of individual WeGovNow components and by interactions between these components and the core features of the platform architecture.

In that sense, the design of the architecture presented throughout this document has been informed by integration requirements expressed in terms of “technology driven usage scenarios”. These were derived through an a priori assessment of technological options generally available for enabling the development of new features which are envisaged to support the anticipated platform users. In particular, the core features of WeGovNow are the result of this preliminary technology driven assessment process.

The outcome of this initial assessment are augmented by dedicated service scenarios and use cases which are currently under development, in terms of a joint effort involving trial site representatives, local stakeholders and the project consortium. These service scenarios are being tailored towards operational tasks to be supported by the WeGovNow platform, involving multiple local actors in staged adoption processes. These service scenarios, and related use cases respectively, are being analysed with a view to identifying service related requirements on the iterative development of the final prototype platform.

In the following subsections, outcome and implications of both strands of work are presented.

## 5.1 Technology driven usage scenarios

The technical driven usage scenarios are based on the experience of technical partners, in particular assessing the general architecture in terms of information flow and protocols between platform components. They were developed under the assumption of achieving a modular and principally extendable platform architecture which nevertheless provides the impression of a single, coherent overall online service platform. In the following subsections, two examples of technicality driven usage scenarios are presented which were considered in defining the general platform architecture.

### 5.1.1 User setup

The user setup includes a set of tasks regarding the registration, activation of the account and collection of the required information from the user. The common platform could collapse most of the tasks within the registration phase, redirecting the user to their profile page to finalise optional information. Since WeGovNow is a modularly composed platform, the task of setting up the user's profile becomes very complex, in particular when it comes to harmonising the different requirements from single components, to the platform management, to its usability and the like. For instance:

- The municipality requiring the platform may have their own requirements about user's profile and verification;
- Each component may require different information from users;
- Propagation of user's preferences toward all components;
- Avoiding duplicate requests of the same information.

The solution implemented in the WeGovNow architecture is a mediation of the four main perspectives we have within this task:

1. The municipality needs to define the rules of using WeGovNow;
2. The component perspective of collecting the required information to provide their functionalities;
3. The requirement of building a modular and extendable platform;
4. The end user who does not fill information in duplicated forms, and requires one single point to manage his/her information.

The solution provided within WeGovNow uses multiple components and strategies such as:

- Agreement on OAuth 2.0 standard for authentication;
- A lightweight centralised user profile and setup system in UWUM, managing shared parameters;

- The registration form in UWUM defined on the municipality requirements;
- An error code for missing user's info in case of cross platform features, with the reference to the component profile manager;
- A common policy for components to retrieve user's settings on fly, for instance notification setups, accessibility setups, language, etc.;
- The reference (link) to each component profile setup as part of the registration of components within WeGovNow;
- A centralised user's profile and settings page in UWUM and a centralised wizard to fill the profile.

### 5.1.2 Data entry

The data entry, in the sense of creating or updating entries, in a simple application implicates the user's capabilities and business login, but in a modular platform it raises specific issues with regard to data interoperability. In WeGovNow any component can create new entry but those entries are most of the time relevant for other applications. For instance, one component can be used to create a proposal and a second one to collect support and opinions. Moreover, the metadata (data structure) should support a consistent presentation of entries across the WeGovNow platform.

In response to these requirements, the WeGovNow architecture has been designed to support:

- Data interoperability in terms of interconnecting component entries and open data, to consolidate during the use of the platform a common dataset;
- Data presentation in terms of providing a uniform presentation of common entries to users.

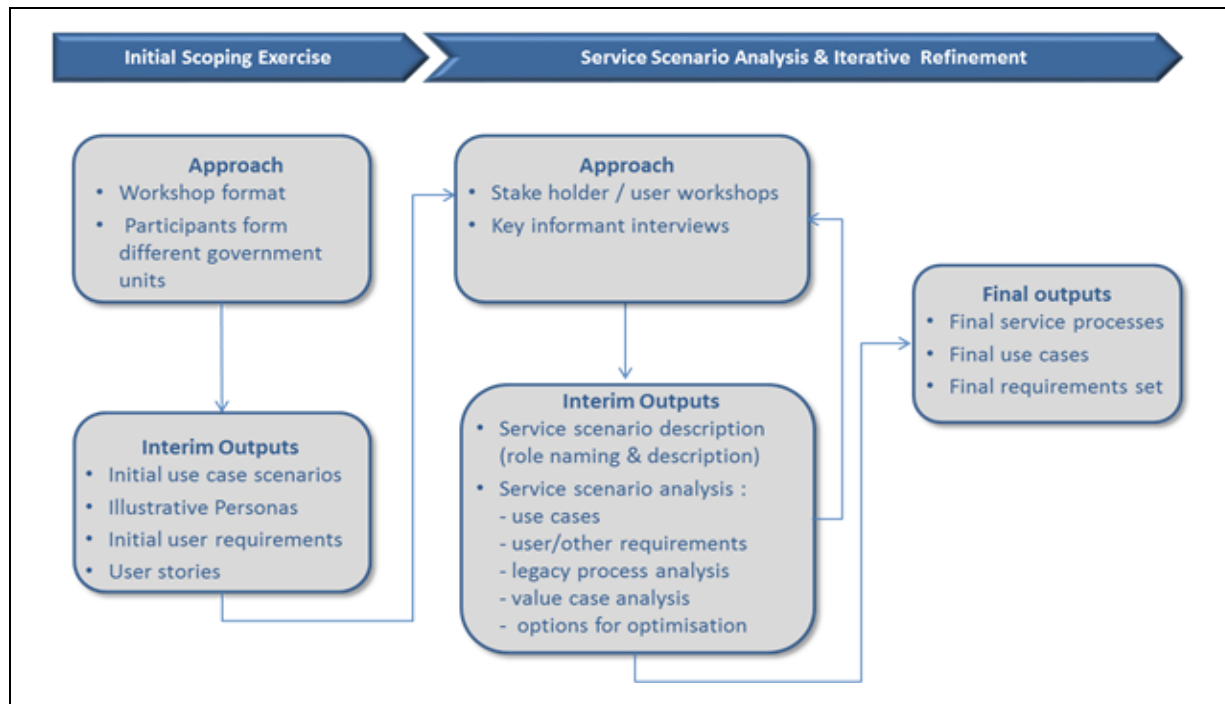
In the first case, a component was defined, InputMap, to enrich the collection of spatial features with explicit references to the common entities collected in OnToMap. In the second case, a translation service was defined which converts the data from their original source to the common ontology defined in OnToMap. The translation service working with the semantic query system makes it possible to make the component entries accessible platform-wise, despite the internal schema.

## 5.2 Service scenarios

Based on a User Centered Design approach, the description of work outlines how the WeGovNow platform's design will be based according to requirements driven by the users' needs. The first set of service scenarios developed with the municipalities as well as dedicated workshops and interviews provided a solid understanding of what each municipality wants to use the platform for within its particular local context (see Exhibit 41 for the WeGovNow use case development approach). These scenarios formed the basis of establishing user requirements, as they provided information about how the municipalities currently work, their internal processes, what they want to achieve through the

WeGovNow platform, as well as process-related requirements on new, WeGovNow-supported public service models that need to be put in place. Along with these service scenarios (see D2.2) the municipalities also provided some descriptive preliminary requirements which were used by UCL to provide a combined list of functional and non-functional requirements (seen Appendix 5.2).

*Exhibit 41: WeGovNow use case development approach*



### 5.3 Connecting requirements and platform development

The work on service scenario development and related use case development is organised in terms of an iterative process enabling a stepwise evolution and prioritisation until the start of the piloting stage, whereby the new WeGovNow services will be provided by the pilot municipalities under day-to-day conditions. As a result, coordination between development tasks and engagement activities is needed to refine proposals, agree on the features to be developed and how these are prioritised and evaluated.

The inputs fed into the platform development cycle can be divided into two components, namely proposals and issues arising from engagement activities at the trial sites and from the assessment of the technical partners. Thus, a synchronisation phase will consist of a mutual exchange and summary of:

- Issues that cannot be solved without intervention of two or more parties;
- Proposals needing to be validated, approved, acquired or tested.

Issues may rise from the technology development and from engagement activities. For instance, a technical issue can be related to the limit of existing technology, timing or cost of a specific requested feature; an issue rising from engagement activities may be a feature

request or a technical problem related to existing features. Issues should be about problems which cannot be solved without other partners, and should be strongly motivated. In other words, “nice to have” requirements should not be considered as issues, but can be relegated to bilateral agreements between partners not requiring central coordination. The MoSCoW method will be applied from the service provider’s perspective (i.e. the municipalities) to assess whether an issue is critical or “nice to have”. MoSCoW is a technique used in management, business analysis, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement - also known as MoSCoW prioritisation or MoSCoW analysis<sup>26</sup>.

Proposals may be both inputs or replies to issues raised in the previous interaction cycle. For instance, a proposal can be a mock-up to be validated or a new feature prototype to be discussed, a protocol about the way to integrate a tool in a real process. Proposals should be about addressing issues or independent initiatives for the sake of the project. Proposals are not requests of new activities, but the offer of a solution: it is not a request for intervention to other partners. A proposal requiring activities from multiple partners should be presented only after an agreement among the involved partners.

A coordination document will be compiled along a cycle of activities and modified along with the interactions among project partners. In other words, a coordination document will provide a snapshot of the current state of a continuous exchange of issues and proposals among trial sites and technical partners based on ongoing engagement activities and technical developments.

## 5.4 Assessment of use cases and user requirements

The service scenarios and user requirements resulting from the engagement activities of stakeholders and end users are being assessed by the technical teams of WeGovNow, in order to identify:

- Priorities of requirements under the light of the adoption of the platform and the service which should be provided;
- Existing/possible solutions within the components;
- Relevance and coherence of the requested features within each component and platform wise.

The assessment is particularly challenging in WeGovNow because the specific nature of WeGovNow platform:

- The complexity of the features provided by the platform does not allow only one solution, moreover the solutions including combination of multiple components must consider their existing pattern of use and technical requisites;

---

<sup>26</sup> Achimugu, P., Selamat, A., Ibrahim, R., & Mahrin, M. N. R. (2014). A systematic literature review of software requirements prioritization research. *Information and software technology*, 56(6), 568-585.

- The complexity of the processes which will involve the use of the platform involving multiple actors at different times may require different scopes and capabilities for each actor, and branch choices.

In WeGovNow we adopted as general approach an analytical framework to rearrange the service scenarios in terms of actors (types), processes and actions in an “assessment grid” (see appendix 5.4). The identification of the overall process and type of actors is used to generalise a set of service scenarios. Actions are evaluated considering the actor and the process phase, identifying the existing component providing the relative feature. The assessment grid is also used to verify with the municipalities if the service can still be provided considering the technical constraints (how an action can be performed) and the pruning (what will be supported).

Actions and requirements are then connected within a requirement grid to support the development, testing and evaluation of the involved features (details in Appendix 5.2).

## 6 Modularity, extendibility and challenges of WeGovNow platform

In a nutshell, WeGovNow is first of all a technical environment to integrate modular components. In general, WeGovNow is extensible with any web-application compliant with the basic requirements (see Chapter 3). The modular nature of WeGovNow poses a challenge in terms of technical solutions for core features and in general to the system architecture.

Each WeGovNow instance (an installation of WeGovNow) has specific configurations regarding the platform styling, the available components, required user profile informations, terms of use, etc. The flexibility required to personalise WeGovNow requires a careful design of the overall system, of the core features and of the cross-component features:

1. The general architecture should work with a minimal set of features, allowing the choice of enabling/disabling components anytime;
2. The cross-component features should be enabled and disabled dynamically as a new component becomes available, since configurations may be changed anytime;
3. The cost of the orchestration of components in WeGovNow must be constant, regardless the number of components;
4. New components should be easily adapted to work within WeGovNow;
5. WeGovNow should be resilient regarding its components: the overall service must be provided regardless the limits or issues of one or more components;
6. Common information must be shared and kept in synch platform wise (e.g. user's preferences).

In the design of WeGovNow, we adopted a set of solutions to address specifically the mentioned desiderata, including in particular:

1. Configurable components in case of the use of existing software and ad hoc small modules (e.g. AreaViewer and LandingPage);
2. Application discovery API provides the current list and parameters of enabled components. It can be used to turn on and off features according to the current global setups;
3. Centralised services to avoid one-to-one connections among components in case of required features (such as search and user profile);
4. Adoption of standards and definition of guidelines
5. Testing and monitoring of core services in worst case scenarios. Scalability of core services;
6. Centralised data sources for local data, user's profile and settings. Technical solutions to integrate application data generated by WeGovNow components.

In general, there are two main aspects to be considered with regard to integration:

- Providing WeGovNow services to users and components;
- Enabling the components to provide their services to other components and users.

The delivery of services and features in both cases needs to be addressed via testing for the correctness and scalability, and monitoring for the availability.

## 6.1 Testing

Testing on the overall integrated WeGovNow platform in terms of usability, functionality and accessibility plays an important role during the development of the project. Testing includes both automated scripts but also user evaluation sessions.

The testing process, from the technical point of view, will run in parallel to software development and each development stage will be tested, validated and verified. Software will be tested on various levels such as;

- **Unit Testing.** The programmer will perform tests on that program unit to ensure it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.
- **Integration Testing.** Even if the units of WeGovNow components are working fine individually, there is a need to find out if the units, integrated together, would also work without errors. For example, argument passing, data enqueueing, etc.

Also, accessibility and usability play a major role in WeGovNow. They have a central place in the User Centered Design development and evaluation. The testing protocol of WeGovNow is presented in a dedicated deliverable (D3.2).

## 6.2 WeGovNow monitoring

The scalability of WeGovNow platform is related to the scalability of core features and to the scalability of modular components. In general, the architecture of core features is design to not be a bottleneck for the platform, for instance, implementing asynchronous protocols and queues. The scalability of components can at most jeopardise a subset of WeGovNow features, but it does not affect the overall functioning of the platform (see also D 3.2).

The monitoring of the WeGovNow platform is addressed considering the common features and the single components. In the first case, the monitoring is part of the management tasks of an instance of WeGovNow. In case of single components and features, the monitoring is the responsibility of individual partners, in case the component is provided as “service”.

During the deployed usage of the platform, ongoing assessment of OpenStreetMap (OSM) and where applicable Public Sector Information (PSI) data incorporated into the platform will be conducted. Using methods identified prior to release (and used in documentation both before and at deployment), reporting will be generated at regular intervals with an additional overview of how the quality of the data has changed over the duration of the project. This evaluative overview will be of utilised to support future deployments beyond the immediate project duration as it will aid the understanding of how the information matures. For example, if the OSM data becomes higher quality through the duration of the project, then in future deployments it may be of benefit to rely more on OSM data after an initial maturity level of the deployment has been reached. Also, the reporting will provide insights as to how frequent the often time consuming update of local OSM databases needs to be performed if such local databases are used.

## 6.3 Connecting with WeGovNow core

The WeGovNow platform has been designed to be principally open towards further extensions. Intercommunication between existing and future components is authorized and coordinated through UWUM, as introduced in section 2.1 of this document. Dynamic client registration (see section 2.1.3) enables application developers to create applications that may be automatically usable for any installation of WeGovNow (or a successor), including future installations that are unknown by the time of application development. In combination with the “client” endpoint (see section 2.1.2), new applications can not only be easily authorized but will be able to automatically discover other applications being part of the WeGovNow cluster (including those being added by the system operator or municipality as well as those being added by individual users of the system). Specification and implementation of the “client” endpoint will be done by LiquidFeedback in close cooperation with UniTo.

## 6.4 Connecting with WeGovNow data

The mapping of heterogeneous data into OnToMap concerns two main types of information:

- **Open Data:** In this case, the data categories defined in the Open Data Source have to be mapped, or added, to the ontology. The mappings are defined between the data representations used by the Open Data Sources and the RDF representation adopted in OnToMap Open Data Store, because they are aimed at translating external data items to Resource Description Framework (RDF) (<https://www.w3.org/RDF/>), RDF triples (Linked Data), to be loaded in OnToMap Open Data Store. As Open Data sources typically use standard data representations, suitable open source mapping tools can be found for doing this work without defining ad-hoc software. For instance, for mapping shape files to RDF data we used GeoTriples (<https://github.com/LinkedEOData/GeoTriples>).
- **Crowdsourced data:** The translation of the data provided by WeGovNow applications to the common format defined by the OnToMap Ontology is done by applying a set of *concept translation rules* (to be defined when the application is integrated in the WeGovNow platform) which specify the correspondence between the data categories of the application and the concepts of the OnToMap Ontology.

Both the OnToMap Ontology and the concept translation rules are specified using declarative languages. This approach supports the integration of new data types without requiring any changes to the core software of the WeGovNow platform, because it consists of revising textual documents. However, it requires understanding the concepts and relations defined in the OnToMap Ontology, in order to be able to accommodate new concepts and/or map categories taking the existing ones into account. We describe the cases to be considered when mapping a new conceptualization to the OnToMap Ontology in the following, focusing on the integration of WeGovNow application data. The case of Open Data is conceptually similar, and only differs with respect to the tools used for translating data to RDF.

- **CASE1: add new concept to OnToMap Ontology**

A new concept C, which belongs to the domain conceptualization to be integrated, has to be added to the OnToMap ontology. For instance, let's suppose that the ontology includes concepts "Hotel" and "Bed&Breakfast", which are subsets of "Accommodation", and we have to add the concept of "RentedApartment". "RentedApartment" has similarities with the other two concepts but represents a new category of information. In this case, the following steps have to be carried out:

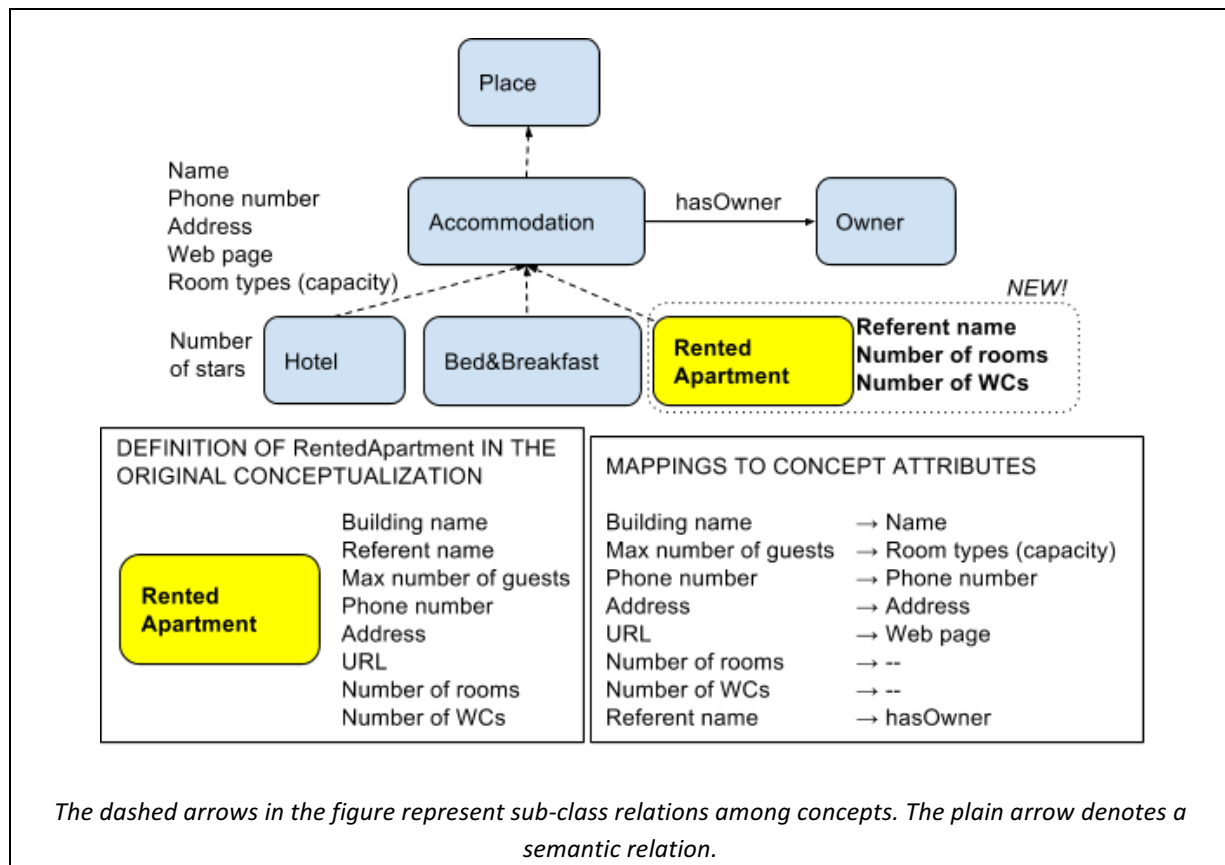
- a. Identify the ontology concept X such that X subsumes C ("RentedApartment") in a taxonomical hierarchy, i.e., the set of instances of C are also instances of X. Note that X must be the "smallest concept" that subsumes C (i.e., the concept with the minimal extension), in order to respect taxonomical relations and minimize the introduction of redundant

information in the ontology. In the above example, X should be “Accommodation”; see the figure below.

- b. Map the attributes of C to the properties defined in X (or inherited by X), whenever possible, by defining the corresponding mapping rules. If C has some attributes that were not previously represented in X, and can be useful to better characterize the instances of X, add such attributes as properties of X, instead of adding them to C. The attributes of C that cannot be mapped to any property of X, nor added to it, are associated to C.
- c. Map the attributes of C to the relations between X and the other concepts of the OnToMap Ontology, by defining the corresponding mapping rules. Relations can be directly associated to C or they could be inherited by C. For instance, let’s suppose that the OnToMap Ontology defines the “hasOwner” relation between concepts “Accommodation” and “Owner” to associate accommodations to their owners. Then, “RentedApartment” inherits the relation. Thus, attribute “Referent name” of “RentedApartment” has to be mapped to this relation.

The following figure shows (Exhibit 42) the revised ontology, with the new concept in yellow for easy recognition. The lower portion of the figure shows the definition of “RentedApartment” used by the WeGovNow application and provides a pictorial representation of the mappings to the format of the OnToMap Ontology.

Exhibit 42: Integration of concept “RentedApartment” into the OnToMap Ontology



- **CASE2: map concept to existing concept of OnToMap Ontology**

In this case, a concept C, belonging to the domain conceptualization to be integrated, can be mapped to a concept Y of the OnToMap Ontology because it describes the same type of information represented by Y. For instance, suppose that the OnToMap Ontology contains concept “Kindergarten” and that a concept “ChildCare” has to be mapped. In this case the attributes of C must be mapped to the (proper or inherited) properties/relations of Y, possibly adding properties to Y if C brings some new features that can be useful to describe the instances of Y. The work to be done is similar to that performed to map attributes in the CASE 1 above, but no concepts are introduced, and thus all the attributes of C have to be mapped to the (proper or inherited) properties/relations of Y.

### Concept translation rules.

The rules for translating the representations of geographical objects provided by WeGovNow applications to the OnToMap Ontology are defined in a JSON object valid with respect to schema [https://ontomap.eu/mapping\\_schema.json](https://ontomap.eu/mapping_schema.json), reported in Exhibit 44 below.

A JSON object (and file) is associated to each WeGovNow application to represent the mappings between the specific application and OnToMap Ontology. The JSON object must contain a field (“application”) representing the application to which the mapping is referred, and a list of concept mapping rules (“mappings”). Each rule contains the name of the concept in the application domain (“app\_concept”) and the name of the corresponding concept in OnToMap Ontology (“ontomap\_ontology”). Moreover, it contains and a list of property mappings, stored in attribute “properties”:

- Each property mapping specifies the name of the property in the application domain (“app\_property”) and the name of the corresponding object property in OnToMap Ontology (“ontomap\_property”).
- In case of properties representing measures (e.g., price, height, weight, size), different WeGovNow applications might use different measure units; e.g., an application might express the price of the items belonging to a category in Euros, another one might use Sterlings. In order to correctly store the reference unit of the user activity events, the property mapping must specify the measure unit of the property to be stored in the log (e.g., Euros, cm., Kg., cm<sup>2</sup>). See “unit” in the schema. Notice that we assume that a single application handles the measure unit of each property consistently; i.e., all the events concerning the property must use the same measure unit.

For instance, the rules (Exhibit 43) for the previous Kindergarten - ChildCare example, used to generate the translation shown in Exhibit 7 on page 22, are reported below.

*Exhibit 43: Rule representing the mapping between the schema of an application App1 and the schema of OnToMap*

```
{
  "application": "App1",
  "mappings":
  [
    {
      "app_concept": "ChildCare",
      "ontomap_concept": "Kindergarten",
      "properties":
      [
        {
          "app_property": "denominazione",
          "ontomap_property": "hasName"
        },
        {
          "app_property": "indirizzo",
          "ontomap_property": "hasAddress"
        },
        {
          "app_property": "telefono",
          "ontomap_property": "hasPhoneNumber"
        },
        {
          "app_property": "retta_mensile",
          "ontomap_property": "hasMonthlyRate",
          "unit": "EUR"
        }
      ]
    },
    {
      "app_concept": "Ristoranti",
      "ontomap_concept": "Restaurant"
    }
  ]
}
```

The following table reports the syntax of the concept translation rules in detail.

*Exhibit 44: JSON schema specifying the concept translation rules to OnToMap Logger format*

**Syntax of the concept translation rules for mapping the data categories of a WeGovNow application to the OnToMap Ontology concepts.**

URL: [https://ontomap.eu/mapping\\_schema.json](https://ontomap.eu/mapping_schema.json)

```
{
  "$schema": "http://json-schema.org/schema#",
  "definitions": {
```

```

"concept_mapping": {
  "type": "object",
  "properties": {
    "app_concept": {
      "type": "string"
    },
    "ontomap_concept": {
      "type": "string"
    },
    "properties": {
      "type": "array",
      "items": {"$ref": "#/definitions/property_mapping"}
    }
  },
  "required": ["app_concept", "ontomap_concept"],
  "additionalProperties": false
},

"property_mapping": {
  "type": "object",
  "properties": {
    "app_property": {
      "type": "string"
    },
    "ontomap_property": {
      "type": "string"
    },
    "unit": {
      "type": "string"
    }
  },
  "required": ["app_property", "ontomap_property"],
  "additionalProperties": false
}
},

"type": "object",
"properties": {
  "application": {
    "type": "string"
  },
  "mappings": {
    "type": "array",
    "items": {"$ref": "#/definitions/concept_mapping"}
  }
},
"required": ["application", "mappings"],
"additionalProperties": false
}

```

### 6.4.1 Look and Feel

As a sum of a number of stand-alone components, the WeGovNow architecture raises the strong need for defining a solution that is capable of creating a coherent user experience in terms of a common “look and feel” across all platform components. The coherency of the platform has multiple reasons:

- Building the identity of the platform;
- Provide an overall good user experience;
- Supporting the “walkthrough” among the features of the platform.

The coherence between independent components is a critical part of the accessibility and usability of the WeGovNow platform, which cannot be achieved by addressing this issue only component wise. The language spoken by the platform must be consistent across all its components to avoid generating confusion and misunderstandings.

In WeGovNow achieving the common “look and feel” is complicated by the modular nature of the platform: existing components have their own consolidated solutions, and future components cannot be strictly controlled. Therefore, we are developing lightweight solutions based on design guidelines to be gradually implemented, to shared core features and modules (see core features in section 2).

In particular, in creating the common look and feel we identified a number of critical points as follows:

- **Error management**, a common set of errors to be handled by any components to be defined. In order to achieve good usability and accessibility, errors will be displayed in a unified and harmonized manner across all components;
- **Propagation of user’s setup** about language, accessibility preferences, notifications thanks to a shared repository within UWUM and the common policy to keep in synchronisation the component and the shared preferences;
- **Design of the web interface**, with the priority to end users. WeGovNow provides through UWUM a style service to retrieve a common stylesheet, including the style setups of the current instance. Moreover, within WeGovNow we adopt the Material Design guidelines (<https://material.io/guidelines/>) and a common repository of design solutions (Design Styleguide, see section 7.5) to converge toward similar layouts. As part of this process, Funka are also modifying the material design guidelines in key areas in order to ensure a better adherence to WCAG 2.0 (AA) requirements. For instance, the colour contrast requirements have been made stricter;
- **Navigation** between WeGovNow features, provided by the NavigationBar (see section 2.4), is included in all components;
- **The cartography**, the geographical source and map theming should be consistent across the platform. We will define a common source of map tiles and when possible use the same web map (see InputMap and AreaViewer in sections 2.9 and 2.10).

In the development of WeGovNow more common issues are expected, we believe that as general approach, the solutions should be replicable by all existing and future components.

Therefore, the solutions will be provided as core features, libraries, documentation of approaches (design patterns, policies) or embeddable web modules. The repository of existing solutions will be an important section of the platform documentation;

## 7 WeGovNow toolbox

Building WeGovNow as common platform for heterogeneous components is being addressed from multiple angles. The system architecture and the core features provide the operational support and protocols to run a WeGovNow instance, but the common issues shared by WeGovNow components required the definition of a common set of tools, datasets, patterns, libraries and documentation. The definition of common solutions to shared issues has multiple advantages within WeGovNow in terms of software reusability, overall quality of the platform, increasing the overall *“look and feel”*, better performances, etc.

In particular, WeGovNow had to address issues regarding:

- Geographical and temporal features of urban data;
- Assessment and integration of urban data (open data and geographical sources);
- Design of web interfaces.

Up to now, the outcome of the shared effort in addressing the common issues resulted in libraries, design patterns, datasets and documentations available to project partners and to the future developers of new components. These are described in the following subsections.

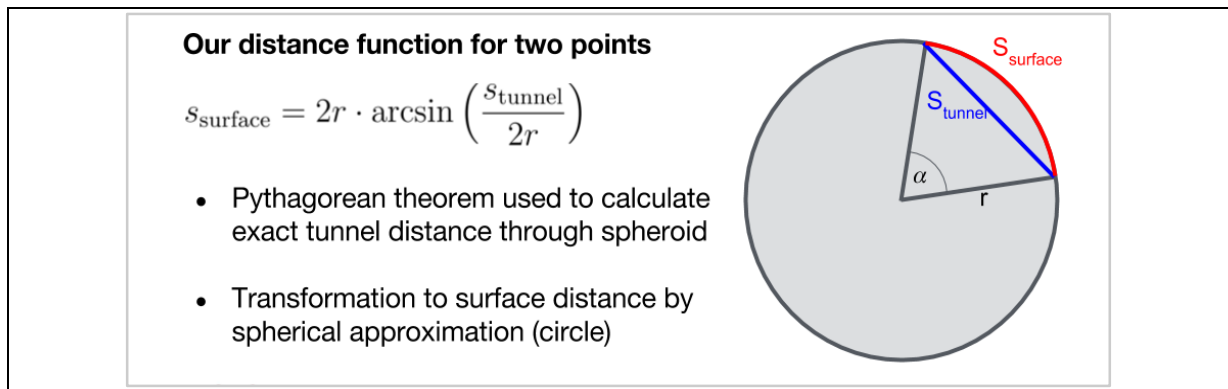
### 7.1 Geographical support

Considering accuracy, performance, dependency and licensing issues, LiquidFeedback decided to implement “pgLatLon”, a spatial database extension for the PostgreSQL object-relational database management system providing geographic data types and spatial indexing for the WGS-84 spheroid.

pgLatLon utilizes a special approximation for distance calculation to circumvent the need for the commonly used Vincenty’s formulae. The general idea to quickly calculate distances on the WGS-84 spheroid is to calculate the exact coordinates of two points in a three dimensional (Euclidean) coordinate system first. Then, the exact tunnel distance can be calculated using the Pythagorean theorem.

After the tunnel distance has been determined, a sphere model of earth is used to transform a tunnel distance into a distance on the surface of earth. The error for small distances tends towards zero. For medium distances, the above formula serves as a good approximation for the surface distance on the WGS-84 spheroid (which is much better than approximating earth as a sphere).

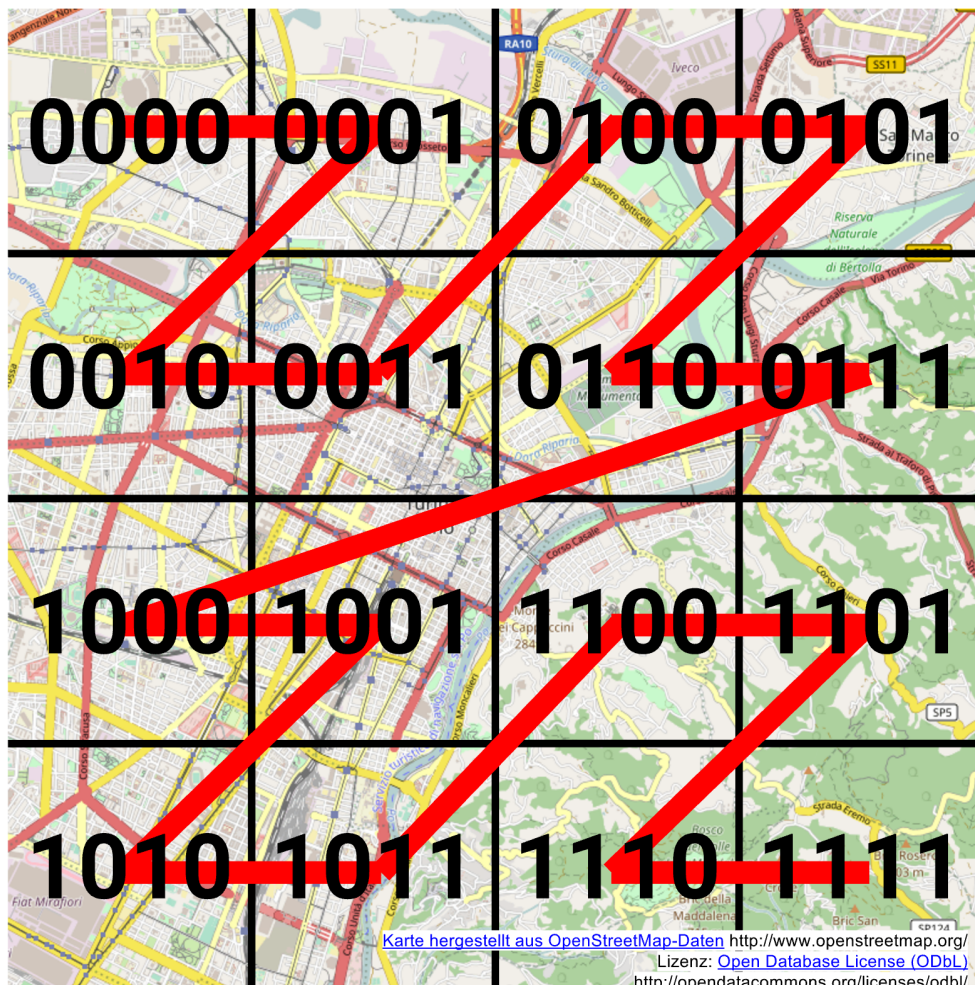
Exhibit 45: Surface distance between two points



There was a challenge with the numerical stability for (nearly) antipodal points. The solution here is to mirror one point through the center of earth, to calculate distance using the mirror point and to subtract the result from  $\frac{1}{2}$  (spherical) earth circumference.

Furthermore, PostgreSQL's GiST indexing framework is used in combination with a space-filling fractal curve to provide an efficient indexing method that operates directly on the spheroid.

Exhibit 46: Fractal indexing using the Z-Order Curve



Optimizations such as logarithmic compression are used to keep index sizes small, which allows applications to store huge amounts of indexable geodata.

As explained in section 3.5 of this document, democratic requirements demand additional functions of geo-spatial databases. As part of “pgLatLon”, a prototype for a “fair distance” function for nearest-neighbour searches of indexed objects was implemented, which fulfills the following properties:

- The function returns an adjusted distance (i.e. distance increased by a penalty) if a geographic object consists of more than one point (e.g. if it covers a small or large area or consists of multiple points entered by the user who created the object).
- The penalty function is continuous (except noise created by numerical integration). That particularly means: small changes in the search point (e.g. center of nearest-neighbour search) cause only small changes in the result.
- For search points far away from indexed objects (i.e. large distances compared to the dimensions of the indexed object), the penalty approaches zero, i.e. the behavior of the “fair distance” function approaches the behavior of the regular WGS-84 surface distance.
- If the indexed object consists of a set of points, the penalty for a search point close to one of those points (closer than half of the minimum distance between each pair of points in the indexed object) is chosen in such a way that the adjusted distance is equal to the distance from the search point to the closest point of the indexed object multiplied by the square root of the count of points of the indexed object.
- If the indexed object does not cover any area (i.e. only consists of points, paths, and/or outlines), and if the center of the nearest-neighbour search overlaps with the indexed object, then the penalty (and thus the result) is zero (i.e. closest possible distance, or best match).
- The integral (or average) of the square of the fair distance value (result of the function) over all possible search points is independent of the indexed object as long as the indexed object does not cover more than a half of earth's surface.

The above properties are believed to aid handling illegit user input of intentionally oversized objects to optimize search results in an unfair manner. At the same time, objects which are legitimately covering a huge area aren't disadvantaged in an unfair manner (due to the last property in the list above). The “fair distance” function is the first instance of a new class of geographic functions which may create a new research field for future projects that combine democratic principles with geospatial data processing.

As a direct outcome of the WeGovNow project, the software “pgLatLon” is already available as open source and will be used by partner UniTo in FirstLife. This also solves potential license issues with PostGIS. For details, refer to LiquidFeedback's “Work report on ‘pgLatLon’, an alternative to PostGIS”. The licensing issues in regard to PostGIS have been summarized in LiquidFeedback's “Work report on extending the LiquidFeedback Core”, subsection 2.2.1 (pages 7 through 9).

## 7.2 Quality assessment

Quality assessment forms an important yet often overlooked component of using data. In particular, when making decisions based on information used it is important that the decision maker is at least aware that there may be errors or inconsistencies in the data that they are using. The purpose of the data quality assessment portion of the WeGovNow project is to provide information to municipalities and where applicable to end users of the platform regarding the quality of data available. In particular, as OSM data which is used in the project (i.e. as a base map) is known to be heterogeneous in nature and has varying levels of completeness, it is important that people are aware of any aspects of it that may affect outcomes of decisions made.

As reported in another deliverable (D1.2), a comprehensive review of existing approaches to data quality assessment was been carried out. It turned out that only intrinsic assessment could be implemented (since quality large scale reference datasets are not available in frame of WeGovNow project).

Prior to the deployment of the platform in the municipalities, the quality of the underlying OpenStreetMap (OSM) data used in base maps and (where feasible) the Public Sector Information (PSI) data obtained for use in the platform will be investigated. On the first level, a general overview of OSM in the pilot regions will be assessed and documented covering aspects such as determined completeness and currency of the data. This assessment will be performed using various methods which have been identified within a literature review and then assessed for appropriateness. These assessments will be applicable for the determination of how “useful” OSM data is in the pilot sites with regards to what could be included in the Open Data portion of the platform.

Using finalised scenarios which will become available from another work strand of the overall project (WP2), more focussed quality assessment will be conducted on appropriate available data from OSM and PSI as a means of informing the municipalities as to what they should be aware of in relation to the data. These assessments will be collated into a report for each municipality covering the spatial extents of the scenarios provided. Such information can be used during the deployment of the platform to aid the municipalities when decisions are to be made based on the platform. For example, if a decision is to be made regarding the placement of a structure based on information provided by users of the service, the report would guide the municipality as to whether the base data identifying the presence of other similar features in the environment is up to date or not. It is envisioned that a number of the assessment methods identified that were used on the OSM dataset in the general quality assessment will be directly transferable to the PSI data based on attributes identified and available in the OnToMap ontology. For example, if the ontology identifies that a timestamp for data creation is available for all “things”, then methods that use the *created* timestamp meta information for the OSM features would also be compatible with the other PSI data in the platform.

The scenario based quality reports will be delivered along with the deployment of the platform and include textual information regarding the quality of the data alongside charts and maps where appropriate. The methods used to portray the information in maps will be determined through a literature review within the field of spatial quality portrayal, and appropriate methods for portrayal evaluated through engagement with various users of the data.

After deployment, the quality of the OSM data in the pilot sites, and any additional PSI data integrated into the platform will be assessed at regular intervals and documented. These assessments will then be used in the final evaluation stage of the project as a means of assessing whether quality of the data has altered during deployment. This evaluation will be useful for future deployments of the platform outside of the project so that other municipalities are aware of how the underlying data changes over time (i.e. if the quality of the OSM data in the municipality is continually increasing over time, then it's usage may become more applicable in future scenario deployments). Such information would also be useful for determining how frequently to refresh OSM data if it is not being directly accessed from the OSM API (i.e. the data is stored in a local database).

### 7.3 Data sources

Within the WeGovNow platform, the OnToMap Ontology is used as:

- central catalogue describing concepts and relations handled by WeGovNow applications. This information corresponds to:
  - the types of information stored in the Open Data handled by OnToMap;
  - the types of information collected by the WeGovNow applications.

The ontology supports multi-linguality in the specification of concepts, that can be associated to different linguistic descriptions for visualization.

- high-level conceptual model for data integration among WeGovNow applications. The OnToMap ontology can be used to map the high-level concepts defined in it with the concepts/categories/data-types/tables/etc. of the domain conceptualizations of individual applications.

The structure, functions and status of the OnToMap Ontology are presented in Appendix 7.3. Further information is available online, in particular when it comes to:

- The latest version of the OnToMap Ontology can be visualized using WebVOWL visualizer, at the following URL:  
<http://vowl.visualdataweb.org/webvowl/index.html#iri=http://ontomap.eu/ontology/>.
- Moreover, it can be downloaded in OWL format at the following URL:  
<http://ontomap.eu/ontology/>.

The OnToMap Ontology takes the following types of information into account:

- Domain conceptualizations adopted by the WeGovNow applications;

- Domain conceptualizations adopted in the Open Data sources integrated into the WeGovNow platform;
- Schema.org, which provides a standard data representation adopted by main search engines, and promises to become a “de facto” standard for the internet;
- INSPIRE European directives for the publication of Open Data;
- W3C PROV model for modeling data provenance.

Specifically, the current version of the Ontology accommodates the following data sources:

- Domain conceptualization adopted by FirstLife WeGovNow application (based on the schemas provided by the FirstLife team). This conceptualization models concepts related, e.g., to places and services;
- Open Data provided by the Municipality of Torino (shapefiles downloaded from the Geoportale del Comune di Torino: <http://www.comune.torino.it/geoportale/>). This data source modeled, e.g., green areas, bicycle paths, services (e.g., cultural ones, as well as businesses);
- Open Data provided by the Municipality of San Donà di Piave (archives of geographical data represented as shape files). This data source modeled various types of businesses, monuments, and services.

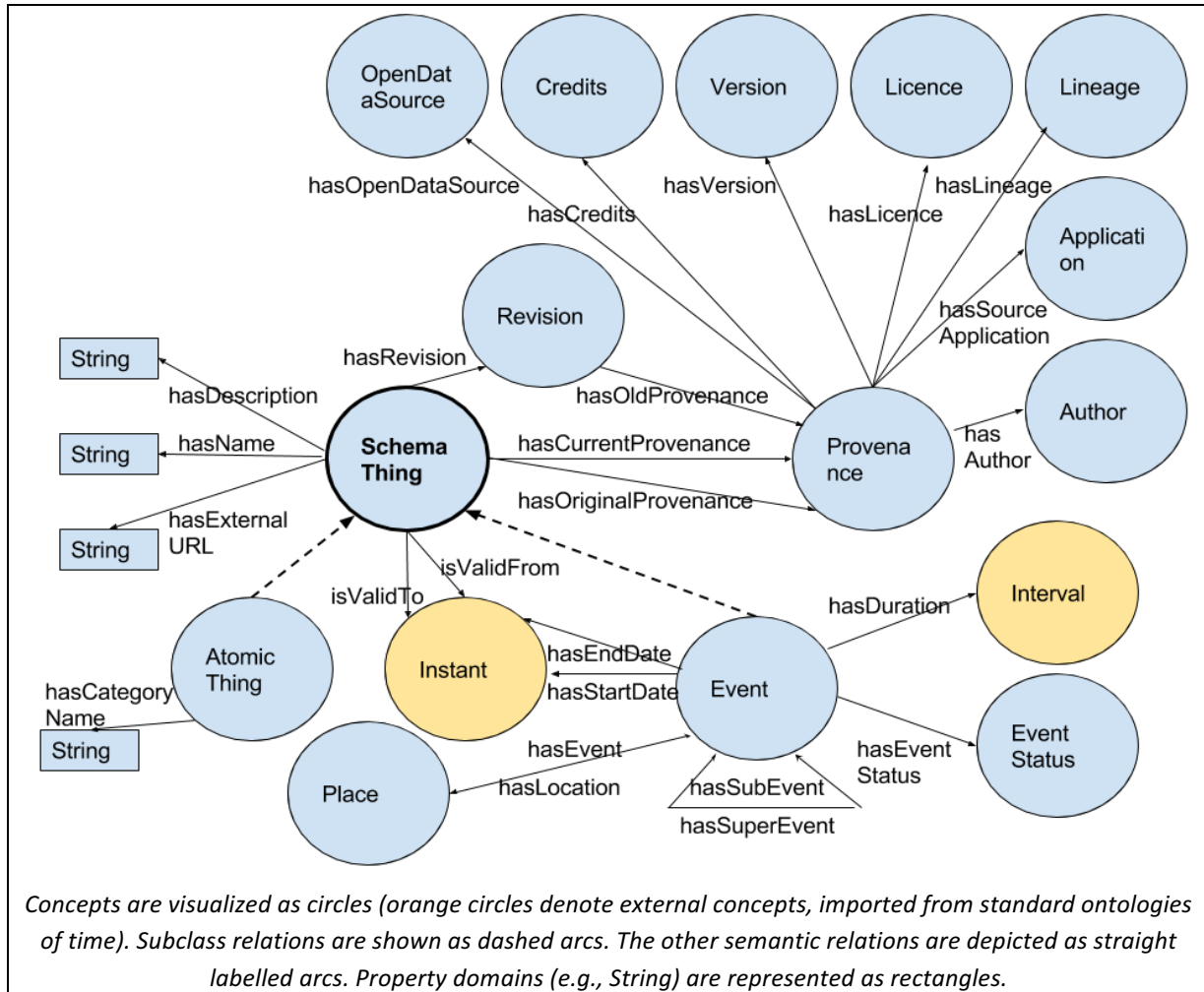
The integration of the Open Data from Southwark is ongoing work.

While the integration of the FirstLife domain conceptualization was rather straightforward, because it represents data categories in a declarative way, the situation was more difficult in the case of the Open Data sources because data was stored in relational tables, shape files, and similar structures, without any associated documentation. Thus, source data categories were reconstructed bottom up, by analyzing the information stored in the shape files, discarding poor data sets (e.g., the tuples in some shape files were almost void; in other cases the names of the attributes of data could not be interpreted), and mapping the rest of the information to the OnToMap Ontology concepts.

In general, the integration of a new concept in ontology (being it derived from an Open Data source, or from an application) might require the introduction of more than one concept in order to take into account the fact that geographical information items might be described at different abstraction levels in the various applications (see the discussion in section 6.3 - mapping data to OnToMap). For instance, the first version of the OnToMap Ontology was defined in conformance with the Open Data provided by the Municipality of Torino, which defined a rich set of categories of services spawning from stores to didactic farms. When we integrated the Open Data provided by the Municipality of San Donà di Piave, we noticed that the representation was based on the Italian ISTAT categories of businesses, which corresponds to a different classification viewpoint. In order to accommodate both cases, we adapted the OnToMap Ontology including concept “Business” as subconcept of “Service”. “Business” represents the typical services reported in the ISTAT categories (and more, if needed), and separates them from the services not strictly related to business, which concern cultural, pedagogical, etc., activities.

The following figure shows a portion of the current version of the OnToMap ontology, focusing on concept “SchemaThing”, the root of the taxonomy concerning geographical information.

*Exhibit 47: Portion of the OnToMap Ontology, focusing on concept “SchemaThing”, depicted in boldface for easy recognition*

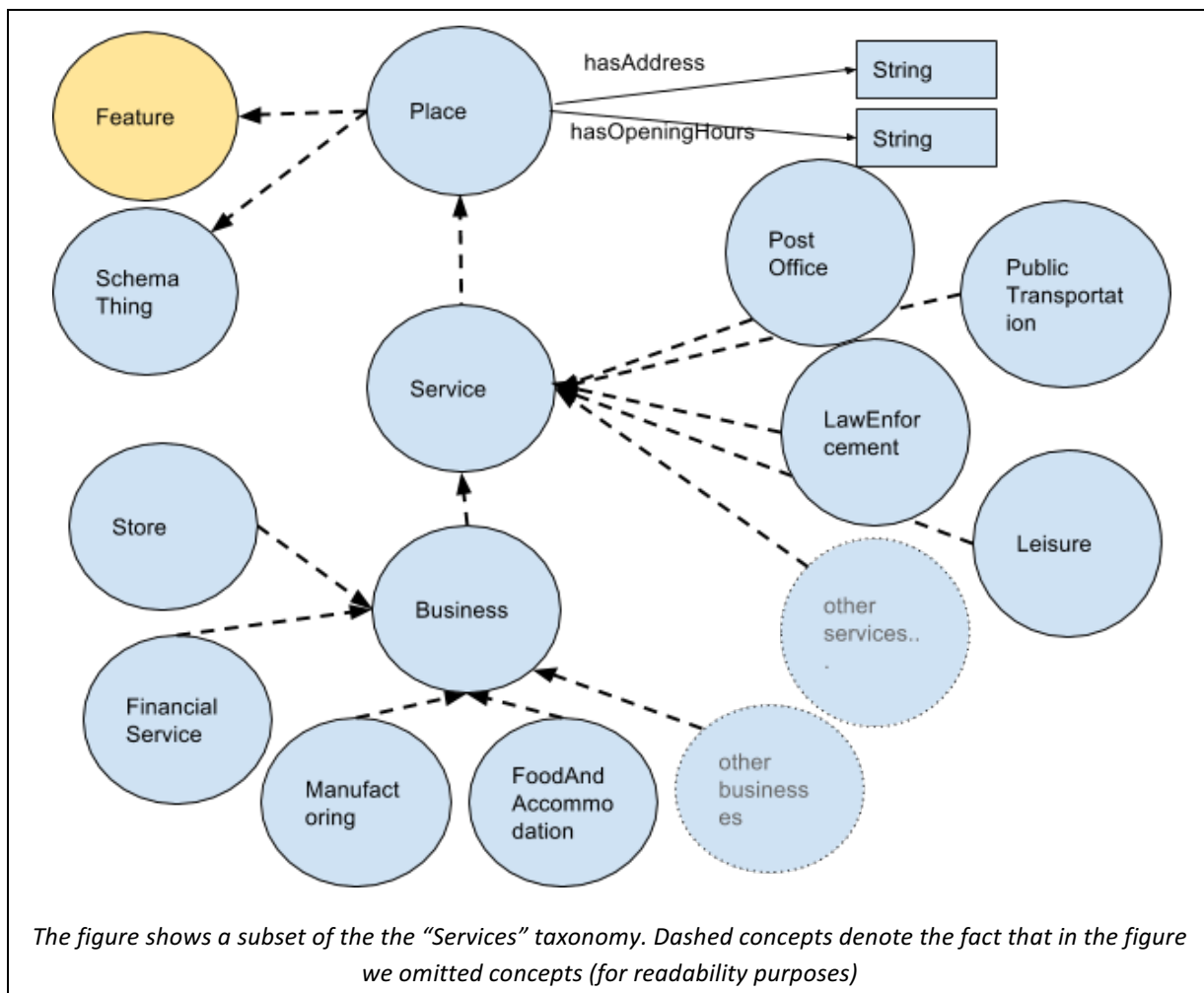


“SchemaThing” specifies the main data structure for geographical information items. It is related to concept “Revision”, in order to comply with the possibility of having multiple versions of data, modified at different times by different users, and to concept “Provenance” in order to support the retrieval of the provenance of the current version of data. In turn, each revision of a geographical data item is related to its own provenance, so that the revision history of data can be reconstructed. The provenance is characterized by several properties, such as the author, version of data, licence of usage, data/application source, and lineage, and makes it possible to specify a single data source as an instance related to multiple geographical objects having the same provenance (e.g., all the open data items belonging to the same dataset). For a detailed specification of provenance, see Appendix 7.3.1.

The OnToMap Ontology supports a more or less tightly coupled data integration, in order to allow both the mapping of stable domain conceptualizations, based on a fixed set of data categories (e.g., FirstLife's one), as well as the integration of dynamic conceptualizations, similar to tag-based systems, which do not impose any restrictions on the data categories to be used to classify information. "SchemaThing" has the following subconcepts:

- "AtomicThing", which represents geographical information based on dynamic data categories. It has a "hasCategory" property which can be used to associate information items to categories, modeled as tags. This approach supports a free type of data tagging, which does not require consistency checks when new categories are added to a domain conceptualization.
- Concepts for the representation of geographic information based on stable domain conceptualizations, which can be mapped to OnToMap Ontology:
  - "Event", which represents events, associated to their temporal extension (see the relations with concept "Instant", which models temporal points), and to the "Places" in which they are held.
  - "Place", which represents all the geographical items having a geometry representing their geographical extension. The subclasses of "Place" are visualized in the following figure. They spawn from various types of green to residential areas, to services, represented by the previously mentioned concept "Service".

Exhibit 48: Portion of the OnToMap Ontology, focused on concept “Place”



## 7.4 Temporal indexing of urban entities

As widely demonstrated by the WeGovNow project, representing urban entities cannot be reduced to the static representation of geographical features. In fact, supporting collaboration, cooperation and planning requires the idea of an evolving world in which the dynamics in term of timing and coordination play a very important role in the success of initiatives. For instance, the scheduling of the occupation of a public square prevents having a strike, a concert and a sport event at the same place at the same time. Urban entities as any social artefacts have a lifecycle, and, furthermore, they are commonly providers of services at local level: the temporal availability of service availability adds an extra temporal dimension in addressing dynamic entities, in particular in case of temporal queries. Times are addressed by almost every commercial technology enabling application to support the stratification of information during time. In particular, most of the major applications for supporting activities in the city enable time queries, time views of data but timings (recurrences and times of activities) are still not fully supported at the city scale.

For these reasons, Unito proposes a light-weighted ready-to-use framework which can be implemented in the most used databases and ORMs based on standard languages. The generic framework, demonstrates how to extend well-known CRON expressions for

representing not only time instants, but also intervals. It shows how to encode CRON expressions by using a 5-tuple bit masks, enabling the storage in standard databases and a very fast retrieval via bitwise queries. Moreover, the framework shows how to extend CRON expressivity in order to represent basic features required in all calendar applications, like:

- Date constraints
- Granularity
- Exceptions
- Occurrences
- Cardinal recurrences

Those features have been realized by introducing new fields in the CRON notation and thus in the CRON bit masks.

Appendix 7.4 reports all details of this approach to the temporal indexing of entities proposed by Unito. The experimental evaluation section demonstrates that a straightforward implementation based on the well-known document database MongoDB can offer very good performances in real world scenarios with thousands of concurrent connections.

## 7.5 Design Guide

In order to converge the current platform components toward a similar outcome and enhance the overall *“look and feel”*, in WeGovNow we adopted Material Design guidelines. The use of guidelines simplifies the coordination and development cost of interfaces but does not solve all possible issues. As coordination and documentation tool, in WeGovNow we are currently defining a project Design Guide.

The Design Guide is a repository of issues and solutions related to the development of the web interfaces of WeGovNow features and components. The design guide is meant to integrate the Material Design guidelines in regards to WeGovNow elements such as buttons, the NavigationBar, fonts, colors, logos, etc. The Design Guide also provides guidelines and best practices for how developers should solve a variety of identified accessibility challenges.

The design guide is meant to be included in the platform documentation, as general indication for future components candidate to be integrated in WeGovNow.

## 8 Conclusions

The design of the WeGovNow overall architecture comes from the unprecedented attempt to combine consolidated softwares in synergy, in order to enable new features, and at the same time to build an environment open to new components and requests from customer municipalities. Addressing those challenges required a relevant effort in analysis of the softwares of WeGovNow in order to identify possible synergies and issues rising from the information flows, navigation and feature combination. Moreover, since the goal of WeGovNow platform is to support processes of we-government, at local scale, it must be configurable accordingly to the local needs. In particular, an instance of the platform must provide the possibility to define the list of enabled components, work with custom components provided by third parties, include styling elements, support the local language platform wide, support the massive use, and finally facilitate the adoption from local stakeholders.

The general architecture of WeGovNow addresses several challenges regarding the requisites of the project regarding:

- The structure and the configuration of the platform;
- The development cycle and design process;
- The software components included in the platform;
- The accessibility guidelines WCAG 2.0 and other requisites for public platforms.

The sum of all requisites within the environment of existing softwares, brought up a complex system of constraints to be considered in defining solutions, protocols and policies:

- Distributed features provided by single or groups of components, keeping their own data and served from different locations;
- Ubiquitous features, resilience of the platform toward single components, which may or may not be enabled within a specific instance of WeGovNow, and which are managed by different teams;
- Asynchronous and scalable features preventing the creation of bottleneck of the platform, information flow compatible with the workload of single components;
- Incremental adoption of solutions and features by the single components, considering the existence of constraints, requirements and development cycles of the single components.

Considering those general approaches, the general architecture addressed the technical challenges of the project.

**Structure and Configuration.** WeGovNow platform is expected to be modular and freely configurable. As general approach, we identified a core set of features we must have to provide the platform and mechanism to support the self configuration of each component accordingly with the specific setup of the platform.

**Development and Design.** The development of new features should be grounded on the requests from the engagement activities, and on the design process open to local stakeholders. In this regards, the development of the architecture was focused on supporting the creation of cross-component features (features resulting from the interaction of multiple components), for the later phase of the development of WeGovNow.

**Components.** Each software have different room for adaptation to the solutions we defined for WeGovNow. Therefore, as general approach the solutions had to be lightweight, based on standards and possibly “optional”. In other words, the partial adoption of the solutions by a component does not affect the general capabilities of the platform, but specific cross-component features only.

This approach does not only considers the specific life cycle of existing components, but enables multiple level of integration for existing and future components.

**Usability and Accessibility.** The modularity of WeGovNow antagonises the overall usability and accessibility of the platform. Every component has its own language, schema, features, interfaces, etc. Therefore, the walkthrough between features of different component will result in confusion from the end user perspective, and in failure in providing the services supported by the platform.

In this regards, to “technical” integration we had to provide a strategy to mitigate the differences between components including common styles, maps, errors, lexicon, etc. The introduction of such mitigation solutions has a very high cost in terms of development and cannot always be applicable at 100% to each component. Therefore, the solutions to increase the overall usability and accessibility of the platform are incremental in the adoption and specific to the specific component in the final result.

The general architecture is completed by a **toolbox** of solutions, a common pool of resources build in parallel to the development of the platform. The toolbox contains libraries, design guidelines, tools, policies and any other solution required to the correct behaviour of the platform or to archive a smooth integration between components.

In conclusion, up to now the overall architecture is defined and most of the core features of WeGovNow are been implemented or are currently under development. Nevertheless, there are many solutions we found that are important to build a quality platform that cannot be implemented in the project time/resource frame, those solutions will be reported for future development of the platform within the scope of the exploitation of WeGovNow.

## 9 Appendices

### ***WeGovNow Core***

- Appendix 2.1.1 - Connecting The Bits
- Appendix 2.1.2 - Work report on Unified WeGovNow User Management development
- Appendix 2.4 - Navigation Bar

### ***WeGovNow Components***

- Appendix 3.3 - Work report on FirstLife API
- Appendix 3.5 - Work report on extending the LiquidFeedback Core
- Appendix 3.7 – Trusted Marketplace mockups

### ***Requirements***

- Appendix 5.2 - User Requirements
- Appendix 5.3 - Assessment Grid
- Appendix 5.4 – Example of Assessment Grid
- Appendix 5.5 – Example of Backlog Grid

### ***Toolbox***

- Appendix 7.1 - Spatial indexing
- Appendix 7.2.1. Portrayal of quality information
- Appendix 7.2.2 - Spatial Data Assessment Tool
- Appendix 7.3 - Details about the OnToMap Ontology
- Appendix 7.4 - Temporal indexing of urban entities

**END OF D3.1**

## **Appendix 2.1.1**

### **Connecting The Bits – Proposal**

## Connecting the bits

---

# Introduction

## Proposal

## Summary

**FLEXIGUIDED**



## Initial situation

---

- **multiple stand-alone applications**
  - featuring emerging technologies
  - mainly web based
  - additional mobile apps
- **all covering different aspects of WeGovernment**

**FLEXIGUIDED**



## Goal: Integration

---

- **create an integrated solution for WeGovernment**
  - provide a unified WeGovNow user interface
- **keep the applications maintained separately**
- **avoid overhead / meta**

**FLEXIGUIDED**



## Goal: Openness

---

- **open source / open specifications**
  - everything to operate a WeGovNow platform
- **recombination in future use cases / developments**
- **3rd party applications**
- **use of standards**

**FLEXIGUIDED**



## Goal: Real World Cases

---

- **implementation of real world cases**
- **production readiness**
  - security
  - data protection

**FLEXIGUIDED**

 **LiquidFeedback**

## Summary of goals

---

- **integration**
- **openness**
- **real world cases**

**FLEXIGUIDED**

 **LiquidFeedback**

## Connecting the bits

---

Introduction

**Proposal**

Summary

**FLEXIGUIDED**

 **LiquidFeedback**

## Proposal

---

**linking user accounts**

**platform navigation**

**look and feel**

**application discovery**

**feature integration**

**FLEXIGUIDED**

 **LiquidFeedback**

## Proposal

---

### **linking user accounts**

meta navigation

look and feel

application discovery

feature integration

**FLEXIGUIDED**

 **LiquidFeedback**

## Linking user accounts

---

- **central account registration**
  - single place to create account and accept terms
- **single sign on**
  - avoid multiple logins

**FLEXIGUIDED**

 **LiquidFeedback**

## Linking user accounts

---

- **unified user identity**
  - a user has the same identity across all applications
- **decentralised storage of additional user data**
  - every application holds it's own user data

**FLEXIGUIDED**



## Single sign on with OAuth 2.0

---

- **very easy token based approach**
- **no additional cryptography needed**
  - all cryptography is handled by TLS (via HTTPS)

**FLEXIGUIDED**



## Single sign on with OAuth 2.0

---

- **open standard**
  - open industry standard
  - used by GitHub, Dropbox, Google, Microsoft, ...
- **extensible for our needs**
  - full standard compliance

**FLEXIGUIDED**



## Single sign on with OAuth 2.0

---

1. **application redirects user to central login page**
2. **users logs in (if not already) and is redirected back**
  - if necessary, user registers account before login
3. **application converts token and checks identity**

**FLEXIGUIDED**



## Single sign on with OAuth 2.0

---

### 1. application redirects user to central login page

User's web browser sends HTTPS request to UWUM:

**GET** <api\_base>/authorization

?response\_type=code

authorization code flow

&client\_id=https://app2.wegovnow.eu/oauth\_redir

client id or endpoint uri\*

&scope=identification,app1read,app2read

requested scope

&state=6b8441515be47e72624597280c3cef24

CSRF protection,  
random per session

\* restrictions apply to avoid covert redirects

**FLEXIGUIDED**

 **LiquidFeedback**

## Single sign on with OAuth 2.0

---

### 2. users logs in and is redirected back

- login
  - using credentials
  - using 3rd party service
    - e.g. Google, Facebook, Microsoft, ...
- if necessary, an account can be created before login
- afterwards, user is redirected back to application

**FLEXIGUIDED**

 **LiquidFeedback**

## Single sign on with OAuth 2.0

---

### 2. users logs in and is redirected back

User's web browser sends HTTPS request to application:

**GET** [https://app2.wegovnow.eu/oauth\\_redir](https://app2.wegovnow.eu/oauth_redir)

CSRF protection

?state=6b8441515be47e72624597280c3cef24

temporary auth token

&code=c48c479116ce7209b974577b36b5b48f

state must be the same as in step 1!

FLEXIGUIDED

 LiquidFeedback

## Single sign on with OAuth 2.0

---

### 3. application converts token and checks identity

Application sends HTTPS request to UWUM:

**POST** <api\_base>/token

check and convert an auth token

?grant\_type=authorization\_code

temporary auth token

&code=c48c479116ce7209b974577b36b5b48f

client id or endpoint url

&client\_id=https://app2.wegovnow.eu/oauth\_redir

must match with step 1

FLEXIGUIDED

 LiquidFeedback

## Single sign on with OAuth 2.0

---

### 3. application converts token and checks identity

```
{  
  "access_token": "08592bd818dace8159006ee640a499a",  
  "token_type": "Bearer",  
  "expires_in": 300,  
  "refresh_token": "3564b6c8a18bf559474d2a544eb5b605",  
  "identification": "Member#12345678"  
}
```

access token for further API calls

lifetime in seconds

refresh token to get a new access token

the user which has been identified

**FLEXIGUIDED**

 **LiquidFeedback**

## Single sign on with OAuth 2.0

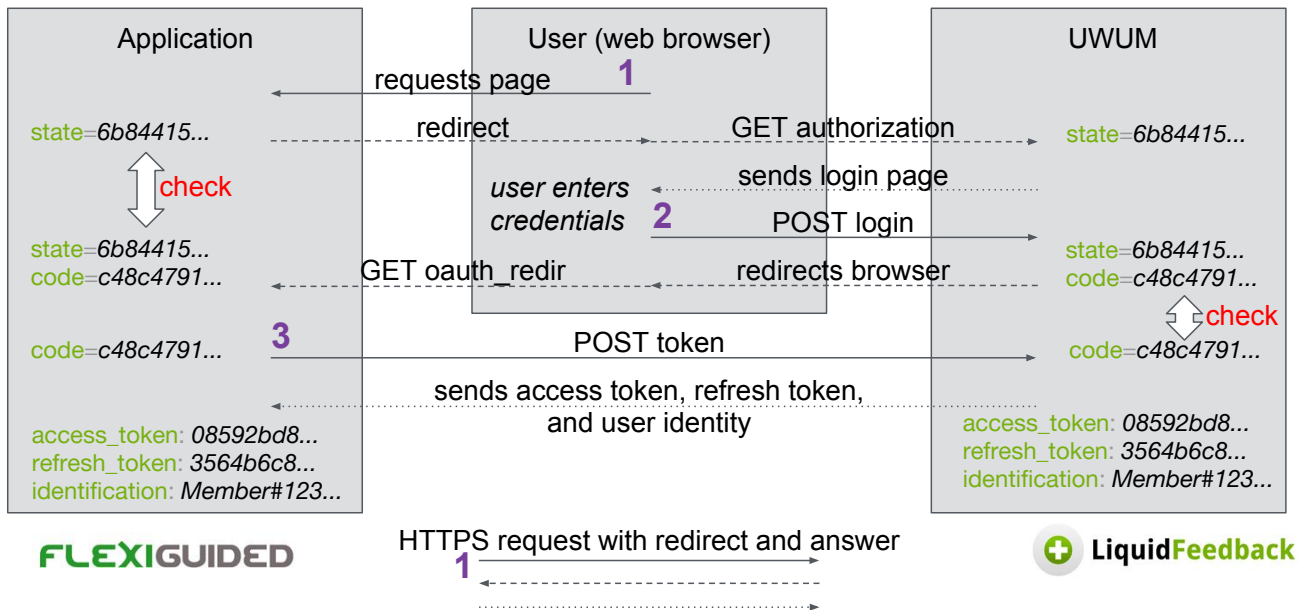
---

1. application redirects user to login page
  - GET authorization (browser → UWUM)
2. users logs in and is redirected back to application
  - GET <oauth\_endpoint> (browser → application)
3. application converts token and checks identity
  - POST token (application → UWUM)

**FLEXIGUIDED**

 **LiquidFeedback**

## Single sign on with OAuth 2.0



## Proposal

linking user accounts  
**platform navigation**  
look and feel  
application discovery  
feature integration

## Platform navigation

- **shared WeGovNow navigation bar**
  - integrated by WeGovNow applications
- **dynamically generated per user**
  - navigation links to other WeGovNow applications
  - user account menu / link

FLEXIGUIDED



## Platform navigation



FLEXIGUIDED



## Platform navigation

---

**GET** <api\_base>/navigation

?format=html

```
{ "template": "<div>
<a href='https://example.com/app1'>Application 1</a>
<a href='https://example.com/app2'>Application 2</a>
<a href='https://example.com/member'>My account
</a></div>" }
```

**FLEXIGUIDED**

 **LiquidFeedback**

## Platform navigation

---

**GET** <api\_base>/navigation

?format=json

```
{ "applications": [
  { "name": "App1", "url": "https://app1.wegovnow.eu/" },
  { "name": "App2", "url": "https://app2.wegovnow.eu/" } ],
  "account_url": "https://app1.wegovnow.eu/profile"
}
```

**FLEXIGUIDED**

 **LiquidFeedback**

## Proposal

---

linking user accounts  
platform navigation  
**look and feel**  
application discovery  
feature integration

FLEXIGUIDED

 LiquidFeedback

## Look and feel

---

- **WeGovNow style guide**
  - keep it simple!
- **configurable platform color theme and font**
  - dynamically adapt application styles
  - matching the CI of use case

FLEXIGUIDED

 LiquidFeedback

## Look and feel

---

**GET** <api\_base>/style

```
{  
  "color": {  
    "background": "#fff", "text": "#000", "link": "#444"  
  },  
  ...  
}
```

**FLEXIGUIDED**

 **LiquidFeedback**

## Proposal

---

linking user accounts  
platform navigation  
look and feel  
**application discovery**  
feature integration

**FLEXIGUIDED**

 **LiquidFeedback**

## Application discovery

---

- **applications can be pre-registered by administrator**
  - can be authenticated by TLS cert or passphrase
- **applications can be registered by users**
- **discovery of pre-registered applications**
- **discovery of user registered applications**

**FLEXIGUIDED**



## Application discovery

---

**GET** <api\_base>/client

?**access\_token**=708592bd818dace8159006ee640a499a

to meet the recommendations stated in RFC 6750,  
the access token can alternatively be provided as HTTP header field

```
{ "clients": [  
  { "name": "LiquidFeedback",  
    "protocol": "org.liquidfeedback",  
    "url": "https://lf.wegovnow.eu/",  
    "registered": true }, ... ]  
}
```

**FLEXIGUIDED**



## Proposal

---

linking user accounts  
platform navigation  
look and feel  
application discovery  
**feature integration**

FLEXIGUIDED



## Feature integration

---

- **deep links to other applications**
- **data embedding from other applications (via API)**
- **view embedding from other applications (via API)**
- **triggering actions at other applications (via API)**

FLEXIGUIDED



## Feature integration: Deep links

---

- **passing user to another application via deep link**
- **multi instance applications**
  - requires passing of WeGovNow instance base URL
- **target application checks login via auth endpoint**
  - only if needed, application enforces login
- **no access tokens need to be passed**

FLEXIGUIDED



## Feature integration: Deep links

---

```
<a href="https://app4.wegovnow.eu/object/1234.html?
wegovnow_endpoint=https://wegovnow.eu/uwum/">
  read more...
</a>
```

FLEXIGUIDED



## Feature integration: Data embedding

---

- **include data from other applications' APIs**
  - obtain access token with necessary scope
  - make API call
    - pass correct access token
    - optionally pass WeGovNow instance base URL
  - other application verifies token
  - other application answers API request

FLEXIGUIDED



## Feature integration: Data embedding

---

### POST <api\_base>/token

?grant\_type=refresh\_token

&refresh\_token=3564b6c8a18bf559474d2a544eb5b605

current refresh token

&scope=identification,app2read

scope needed for API request

{ "access\_token": "e8e4a54a0b6ef3c789842b155b18b4ab",

access token for API calls

"token\_type": "Bearer",

life time of access token in seconds

"expires\_in": 300,

"refresh\_token": "3564b6c8a18bf559474d2a544eb5b605" }

optionally new refresh token

FLEXIGUIDED



## Feature integration: Data embedding

**POST** <api\_base>/validate

?access\_token=e8e4a54a0b6ef3c789842b155b18b4ab

Received access token

```
{  
  "scope": "identification,app2read",  
  "identification": "Member#12345678"  
}
```

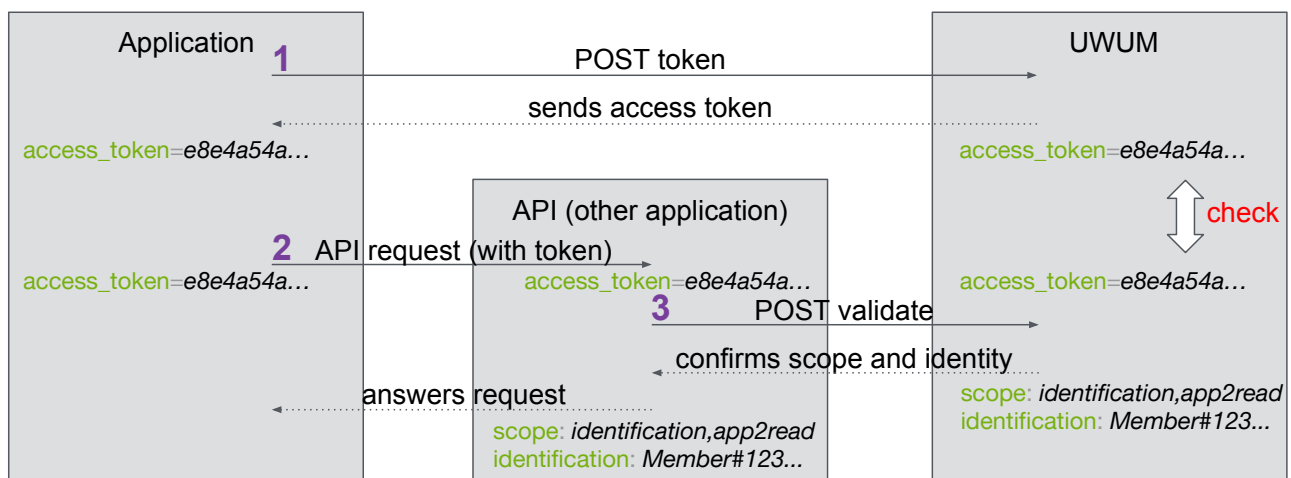
Confirmed scope

Confirmed user identity

**FLEXIGUIDED**

 **LiquidFeedback**

## Feature integration: Data embedding



**FLEXIGUIDED**

HTTPS or API request with answer  
1

 **LiquidFeedback**

## Feature integration: View embedding

---

- include a pre-rendered view from other application
- same OAuth workflow as shown before
  - consumer requests a token: POST <api\_base>/token
  - consumer passes token with a request to provider
  - provider checks the token: POST <api\_base>/validate
  - provider provides

**FLEXIGUIDED**



## Feature integration: View embedding

---

```
<iframe src="https://app2.wegovnow.eu/api/map
?embed=iframe&x=1234&y=2345&zoom=3
&wegovnow_endpoint=https://wegovnow.eu/
&access_token=e8e4a54a0b6ef3c789842b155b..."></iframe>
```

```
<script src="https://app2.wegovnow.eu/api/map
?embed=js&x=1234&y=2345
&wegovnow_endpoint=https://wegovnow.eu/
&access_token=e8e4a54a0b6ef3c789842b155b..."></script>
```

**FLEXIGUIDED**



## Feature integration: Trigger actions

---

- **trigger actions of other application**
  - by making API calls
- **same OAuth workflow as shown before**
  - consumer requests a token: POST <api\_base>/token
  - consumer passes token with a request to provider
  - provider checks the token: POST <api\_base>/validate
  - provider provides

FLEXIGUIDED



## Feature integration

---

- **POST token**
  - (fetch access token with needed scope)
- **data and view embedding**
- **action triggering**
- **POST validate**
  - (validate an access token)

FLEXIGUIDED



## Summary proposal

---

linking user accounts  
platform navigation  
look and feel  
application discovery  
feature integration

**FLEXIGUIDED**

 **LiquidFeedback**

## Connecting the bits

---

Introduction  
Proposal  
**Summary**

**FLEXIGUIDED**

 **LiquidFeedback**

## Summary of goals

---

- integration
- openness
- real world cases

**FLEXIGUIDED**

 **LiquidFeedback**

## Goal: Integration

---

- **create an integrated solution for WeGovernment ✓**
  - provide a unified WeGovNow user interface
- **keep the applications maintained separately ✓**
- **avoid overhead / meta ✓**

**FLEXIGUIDED**

 **LiquidFeedback**

## Goal: Openness

---

- **open source / open specifications** ✓
  - everything to operate a WeGovNow platform
- **recombination in future use cases / developments** ✓
- **3rd party applications** ✓
- **use of standards** ✓

**FLEXIGUIDED**

 **LiquidFeedback**

## Goal: Real World Cases

---

- **implementation of real world cases** ✓
- **production readiness** ✓
  - security
  - data protection

**FLEXIGUIDED**

 **LiquidFeedback**

## Summary

---

- **6 central API interfaces to connect the bits**
  - GET authorization
  - POST token
  - POST validate
  - GET client
  - GET navigation
  - GET style
- **everything else is done bilaterally**

**FLEXIGUIDED**

 **LiquidFeedback**

## **Appendix 2.1.2**

Work report on Unified WeGovNow  
UserManagement (UWUM) development

# Work report on Unified WeGovNow User Management (UWUM) development

Jan Behrens, Axel Kistner, Andreas Nitsche, Björn Swierczek

2016-12-12

© 2016 FlexiGuided GmbH, Berlin

## 1 Presentation of UWUM in Berlin

A first draft of UWUM has been presented in the kick-off meeting “Connecting The Bits” on April 14, 2016 in Berlin. The overall idea was to build a single-sign-on (SSO) solution on OAuth 2.0’s Authorization Code<sup>1</sup> flow.

For access tokens, the use of bearer tokens<sup>2</sup> was proposed. Furthermore, it was agreed on that TLS is to be used to secure all communication between UWUM and other components.

In addition to single-sign-on, UWUM’s capabilities were planned to include:

- a style endpoint, which allows applications to retrieve style information (e.g. a color scheme),
- a navigation endpoint, which allows applications to incorporate a common navigation bar into their user interfaces, and
- a service discovery endpoint, which allows applications to retrieve a list of other applications within the system and their capabilities/protocols.

This way, WeGovNow is designed to be a modular system that may be extended with different services which are all connected through UWUM.

It was agreed that UWUM will be implemented by LiquidFeedback such that it is possible to use synergetic effects between the necessary creation of an API for LiquidFeedback and the newly created features required by UWUM.

---

<sup>1</sup>See <https://tools.ietf.org/html/rfc6749#section-1.3.1> for a short overview on the Authorization Code flow and <https://tools.ietf.org/html/rfc6749#section-4.1> for a detailed description.

<sup>2</sup><https://tools.ietf.org/html/rfc6750>

## 2 Authentication and Authorization

For reasons of interoperability and security, we aimed to create an implementation that is fully compliant with RFC 6749.<sup>3</sup> In this section, the extensions necessary in addition to that document will be explained below. All functionality has been implemented by the time of publishing this work report except where otherwise noted.

### 2.1 Roles

RFC 6749 defines several roles in subsection 1.1.<sup>4</sup> The UWUM component as implemented by LiquidFeedback takes the role of the “authorization server”. Other WeGovNow components will take the role of “clients” but may also act as “resource server” for other components.

### 2.2 Choice of protocol flow

UWUM requires the Authorization Code flow<sup>1</sup> for secure user authentication, i.e. when used for single-sign-on (SSO). (Note that subsection 10.16 in RFC 6749 explains why the Implicit flow<sup>5</sup> as defined by OAuth 2.0 is *not* suitable for secure user authentication.<sup>6</sup>)

The Implicit flow<sup>5</sup> is still supported for clients which only require authorization but do not rely on secure user authentication (e.g. pure JavaScript clients which access other components but do not store themselves any resources which would need to be protected by SSO).

### 2.3 Types of clients

RFC 6749 distinguishes between “confidential clients” (which are capable of secure client authentication, e.g. by maintaining confidentiality of their client credentials) and “public clients” (which are incapable of secure client authentication). UWUM requires all clients which use OAuth 2.0’s Authorization Code<sup>1</sup> flow (and thus receive long-lasting refresh tokens) to be capable of secure authentication; i.e. every use of the token endpoint (see subsections 2.7 and 2.8) will require client authentication (except when an access token scope downgrade

---

<sup>3</sup><https://tools.ietf.org/html/rfc6749>

<sup>4</sup><https://tools.ietf.org/html/rfc6749#section-1.1>

<sup>5</sup>See <https://tools.ietf.org/html/rfc6749#section-1.3.2> for a short overview on the Implicit flow and <https://tools.ietf.org/html/rfc6749#section-4.2> for a detailed description.

<sup>6</sup><https://tools.ietf.org/html/rfc6749#section-10.16>

is performed, see subsection 2.14). The use of “public clients” is only supported for those clients which utilize the Implicit<sup>5</sup> flow because these clients will not handle any long-lasting tokens.

## 2.4 Client registration

Client registration is mentioned in section 2 of RFC 6749, even though the standard explicitly states that “the means through which the client registers with the authorization server are beyond the scope of [the] specification”.<sup>56</sup> UWUM provides two methods of client registration:

- registering clients through the municipality (or their technical administration) or an organization running a particular installation of WeGovNow,
- registration of any other (“dynamic”) client on a per-user basis by each user who wishes to use that client to access WeGovNow (machine accessibility).

These two registration methods are described in the following two subsections respectively.

### 2.4.1 Clients approved by the municipality

Clients approved by the municipality authenticate through TLS (X.509) certificates which are signed by the municipality or a certificate authority acting on their behalf. For example, the operator of the UWUM server could issue a certificate to the operator of each respective client. Furthermore, the operator of the UWUM server configures a list of automatically granted access scopes<sup>7</sup> for the particular client (not every client has the same automatically granted access scopes, e.g. some clients might not require voting rights). Any other access scope may be granted on a per-user basis by the respective end-user or be disallowed by the municipality for a particular client (through white or black lists).

This results in the following information being stored per client:

- name of client,
- OAuth 2.0 client identifier (`client_id`),
- redirect URI(s)<sup>8</sup>,
- common name (CN) of the TLS certificate,

<sup>7</sup><https://tools.ietf.org/html/rfc6749#section-3.3>

<sup>8</sup>See <https://tools.ietf.org/html/rfc6749#section-3.1.2> for redirection URIs. One redirect URI is the default redirect URI, other redirect URIs may be selected through the `redirect_uri` parameter, see: <https://tools.ietf.org/html/rfc6749#section-4.1.1>

- automatically granted scopes<sup>7</sup>,
- white list of scopes (optional),
- black list of scopes (optional, i.e. may be empty).

### 2.4.2 Dynamic clients

For the sake of machine accessibility, it would be nice to allow unregistered clients. Unfortunately, OAuth 2.0 requires some sort of client registration (at least) for the following security reasons:

- allowing capability to authenticate a client,<sup>9</sup> in order
  - to avoid refresh token abuse by a third party in case of accidentally exposed refresh tokens,<sup>10</sup>
  - to avoid authorization code abuse (which could expose access and refresh tokens to a malicious 3<sup>rd</sup> party) in case of exposed authorization codes,<sup>11</sup>
- restriction of choice of the redirect URI<sup>12</sup>, in order
  - to avoid redirection URI manipulation,<sup>13</sup>
  - to avoid open redirector attacks.<sup>14</sup>

In order to be able to provide an open platform, however, it should still be possible to use clients which have not been explicitly approved by the operator of the WeGovNow platform. Assuming there will be more than one WeGovNow installation (e.g. run by different municipalities, each operating their own system), this is necessary in order to enable third parties to provide generic clients that can be used by *any* WeGovNow platform, even those not known to the operator of the client.

Consequently, registration of these clients should happen dynamically without further human interaction.<sup>15</sup> This requires to automatically establish a channel

---

<sup>9</sup>See <https://tools.ietf.org/html/rfc6749#section-2.3> and <https://tools.ietf.org/html/rfc6749#section-10.1>

<sup>10</sup><https://tools.ietf.org/html/rfc6749#section-10.4>

<sup>11</sup><https://tools.ietf.org/html/rfc6749#section-10.5>

<sup>12</sup><https://tools.ietf.org/html/rfc6749#section-3.1.2>

<sup>13</sup><https://tools.ietf.org/html/rfc6749#section-10.6>

<sup>14</sup><https://tools.ietf.org/html/rfc6749#section-10.15>

<sup>15</sup>We assume that every user of WeGovNow is legally entitled to use any client of his or her choice to access his or her data and to perform actions. In cases where a particular operator of LiquidFeedback (e.g. a municipality) wants to decline this right, the use of dynamic clients could be disabled.

of trust between the client and the UWUM server through secure authentication. UWUM relies on the following mechanism to archive secure authentication of a dynamic client:

- a dynamic client is only referenced by its domain, and
- at the choice of each client, registration is performed either
  - by adding a certain entry to the domain's DNS zone<sup>16</sup> or
  - temporarily through a REST API call to the UWUM server with a client-side TLS (X.509) certificate issued to the respective domain and signed by a publicly trusted certificate authority (e.g. "Let's Encrypt")<sup>17</sup>.

Taking into account that it cannot be outruled that TLS certificates could accidentally be exposed to a malicious 3<sup>rd</sup> party and considering that there might be at least one publicly trusted CA which is vulnerable to a state-level attack,<sup>18</sup> we restrict the redirection URI<sup>12</sup> to the following static path on the web server's root level:

`/liquidfeedback_client_redirection_endpoint`

This repels any attempts of "authorization code redirection URI manipulation" as explaiend in subsection 10.6 of section 10 ("Security considerations") of RFC 6749 ("The OAuth 2.0 Authorization Framework")<sup>13</sup> even in cases where dynamic client registration could be forged.

Any client that cannot follow the above redirection URI convention must be registered by the municipality or organization running a particular installation of WeGovNow (see subsection 2.4.1).

As an additional security mechanism, the dynamic registration is always done for a set of access token scopes<sup>7</sup> to be used with a particular OAuth 2.0 flow. Thus a client's redirection endpoint registered for the Authorization Code flow cannot be used by the Implicit flow or vice versa unless the registration is broadened accordingly.

---

<sup>16</sup>A TXT DNS resource record needs to be added to the subdomain "\_liquidfeedback\_client" of the respective domain which must include a so-called magic string (namely "dynamic\_client\_v1") as first entry.

<sup>17</sup>The operator of LiquidFeedback is therefore required to decide on a list of trusted CA's. Many operating systems already ship with such a list of root certificates.

<sup>18</sup>Note that similar security considerations also apply to DNS and the risk of DNS cache poisoning or similar attack vectors. This could, however, be fixed by DNSSEC such that future versions of UWUM might lift the described restrictions for domains which are cryptographically secured.

The operator (e.g. a municipality) may still decide to disallow the use of non-approved (dynamic) clients completely. This would, however, limit machine accessibility and render the platform less open for extensions and unforeseen use cases. An appropriate configuration option will be provided which can also be used to limit the access token scope of dynamic clients (using a white or black list).

Unless dynamic clients are entirely disabled, an additional security warning will be displayed to the user when authorizing such a client. The user will be requested to verify that:

- the client domain is trustworthy,
- the client domain is used to host a legit application to access LiquidFeedback,
- the spelling of the domain name (whose client is going to be authorized) is correct,
- the granted scope of access (access token scope) is intended by the user.

Clients which want to avoid these warnings must be approved by the municipality or organization that is operating the LiquidFeedback system (see subsection 2.4.1).

## 2.5 Access token types

As previously mentioned, bearer tokens<sup>2</sup> as defined in RFC 6750 will be used as access tokens. Therefore, the access token type (`"token_type"`)<sup>19</sup> returned by UWUM is always set to `"bearer"`.

## 2.6 Access token scopes

The following set of generic<sup>20</sup> access token scopes<sup>7</sup> has been specified:

**authentication:** Authenticate the current user by reading its unique static ID and current screen name.

---

<sup>19</sup><https://tools.ietf.org/html/rfc6749#section-7.1>

<sup>20</sup>Application specific scopes could be introduced if they turn out to be necessary in the future. It would also be thinkable for dynamic clients acting as a resource server to provide a set of application specific scopes as part of their registration. Further security analysis would be required for such an extension. See also subsection 5.8 for considerations on generic versus application specific scopes.

**identification:** Identify the current user by reading its unique identification string. Automatically implies scope "authentication".

**notify\_email:** Read the notification e-mail address of the current user.

**read\_contents:** Read any user generated content (without authorship, ratings and votes).

**read\_authors:** Read the author names of user generated content (author's static ID and screen name).

**read\_ratings:** Read ratings (see scope "rate" below) by other users.

**read\_identities:** Read the identities (identification strings) of other users.

**read\_profiles:** Read the profiles of other users (e.g. phone number, self-description, etc).

**post:** Post new content.

**rate:** Rate user generated content (e.g. thumbs up/down, "+1", support an initiative, rate a suggestion).

**vote:** Finally vote for/against user generated content in a decision (e.g. vote on an issue in LiquidFeedback)

**profile:** Read profile data of current user (e.g. phone number, self-description, etc).

**settings:** Read current user's settings (e.g. notification settings, display contrast, etc).

**update\_name:** Modify user's screen name.

**update\_notify\_email:** Modify user's notification e-mail address.

**update\_profile:** Modify profile data (e.g. phone number, self-description, etc).

**update\_settings:** Modify user settings (e.g. notification settings, display contrast, etc).

Note that any of these scopes can also be suffixed with "\_detached" to request the scope for usage also when the user is not logged in (which will be explained in subsection 2.9).

## 2.7 User authentication (single-sign-on)

OAuth 2.0 by itself is not suitable for user authentication. Both the Authorization Code flow<sup>1</sup> and the Implicit flow<sup>5</sup> can be extended to provide user authentication and thus allow to implement a single-sign-on (SSO) system. Because the Implicit flow would require additional security mechanisms to be implemented at client side (where bad implementations result in security vulnerabilities),<sup>6</sup> UWUM extends the Authorization Code flow for the purpose of implementing an SSO solution as described in the following.

In order to protect against authorization code substitution attacks, the UWUM server checks the OAuth 2.0 client identity before accepting an authorization code.<sup>21</sup> This is both a requirement stated in subsection 4.1.3 of RFC 6749 (“The OAuth 2.0 Authorization Framework”)<sup>22</sup> and a recommended countermeasure to avoid authorization code substitution attacks in subsection 4.4.1.13 of RFC 6819 (“OAuth 2.0 Threat Model and Security Considerations”)<sup>23</sup>.

The Access Token Response<sup>24</sup> of the OAuth 2.0 Authorization Code flow gets extended with the field “member\_id” which returns the LiquidFeedback member ID of the signed-in user. OAuth 2.0 clients not aware of this extension are requested to ignore this field as stated in subsection 5.1 of RFC 6749.<sup>25</sup> Nonetheless, these clients may still pass the returned access token to the validate endpoint (see next section) in order to determine the member\_id of the user who has logged in.

## 2.8 Endpoints

RFC 6749 defines two endpoint URLs at the authorization server side: the “authorization endpoint”<sup>26</sup> and the “token endpoint”<sup>27</sup>. These are defined as follows:

- [https://server\\_name/api/1/authorization](https://server_name/api/1/authorization) (GET)
- [https://server\\_name/api/1/token](https://server_name/api/1/token) (POST)<sup>28</sup>

Note that a base path may be appended to the *server\_name* component if applicable.

---

<sup>21</sup>Note that, if the client is authenticating with the UWUM server, the *client\_id* parameter can be omitted by the client when accessing the token endpoint (see next footnote).

<sup>22</sup><https://tools.ietf.org/html/rfc6749#section-4.1.3>

<sup>23</sup><https://tools.ietf.org/html/rfc6819#section-4.4.1.13>

<sup>24</sup><https://tools.ietf.org/html/rfc6749#section-4.1.4>

<sup>25</sup><https://tools.ietf.org/html/rfc6749#section-5.1>

<sup>26</sup><https://tools.ietf.org/html/rfc6749#section-3.1>

<sup>27</sup><https://tools.ietf.org/html/rfc6749#section-3.2>

<sup>28</sup>The server name for the token endpoint may differ for those requests where TLS client certificates are used. See subsection 5.2 for explanation.

RFC 6749 does not specify any method for a resource server to “ensure that an access token presented to it by a given client was issued to that client by the authorization server”.<sup>29</sup> Therefore, an additional validation endpoint has to be specified:

- `https://server_name/api/1/validate` (POST)

The validation endpoint does not require any parameters except the access token (bearer token) to be passed using the mechanisms described in section 2 of RFC 6750.<sup>30</sup> It returns a JSON object with the following fields:

- `scope`: a space separated list of scopes<sup>7</sup> associated with the access token (with any “\_detached” suffix stripped off, see next subsection 2.9),
- `member_id`: an integer set to the id of the user who logged in,
- `logged_in`: a boolean set to false if the user has meanwhile logged out.

Note that the scope of an access token may change when the user logs out. This is explained in the following subsection 2.9. Subsection 2.12 will pick up the issue of user logout again.

There may be situations where an OAuth 2.0 client wants to check whether a user is currently logged in without actually forcing the user’s web browser to perform a login if no user was logged in. To provide this functionality, a 4<sup>th</sup> endpoint (also out of scope of the OAuth 2.0 specification) is added at the authorization server side:

- `https://server_name/api/1/session` (POST)

This “session” endpoint can be accessed directly by a user’s web browser (through a script performing a CORS<sup>31</sup> HTTP request with credentials). Its usage is further explained in subsection 2.10.

## 2.9 Binding lifetime of access and refresh tokens to a users web session by default

Access tokens have an expiry time after which they will be invalidated.<sup>32</sup> In addition to the maximum access token lifetime returned in the Access Token Response,<sup>24</sup> UWUM additionally limits the lifetime of both access tokens and

<sup>29</sup>See section 10.3 of the RFC: <https://tools.ietf.org/html/rfc6749#section-10.3>

<sup>30</sup><https://tools.ietf.org/html/rfc6750#section-2>

<sup>31</sup>Cross-origin resource sharing, see <https://www.w3.org/TR/cors/>

<sup>32</sup><https://tools.ietf.org/html/rfc6749#section-5.1>

refresh tokens to the user's web session at the LiquidFeedback (UWUM) server by default (i.e. if the user logs out, the access tokens and refresh tokens will be immediately invalidated).

Some clients, however, require access longer than the user's login session. For this purpose, access token scopes<sup>7</sup> with the suffix “\_detached” may be requested (e.g. “vote\_detached” instead of “vote”). Whether an application may request these scopes (as well as which scopes may be requested for detached access) depends on the configuration for the particular client, or – in case of dynamic clients – on the configuration for all dynamic clients. An access or refresh token that contains only detached scopes will not be invalidated on user logout. Access tokens, however, will still be invalidated when their expiry time (as denoted by the “expires\_in” field in the Access Token Response<sup>24</sup>) has elapsed, in which case a refresh token must be used to obtain a new access token. Access and refresh tokens which contain both detached and non-detached scopes will only have their non-detached scopes removed on user logout instead of being invalidated completely.

Other than the behavior described above, the “\_detached” scopes behave as any other scope for the authorization<sup>26</sup> and token<sup>27</sup> endpoint. Only the validation endpoint (“api/1/validate”) will strip the suffix “\_detached” from the scope field in its response because it doesn't matter for a validating resource server whether a scope has been granted detached from a web session or not.<sup>33</sup>

Even if the token lifetime is bound to the web session (i.e. when only non-detached scopes are requested), a user's logged in web browser may still automatically re-authorize the client whenever he or she is logged in at UWUM and visits the client's website. If such a client was authorized by the user, the permission can be revoked by the user at any time using a designated configuration dialog provided by the UWUM server.

## 2.10 Checking user login without triggering a login

An interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechanism on its own.<sup>34</sup> UWUM therefore provides an additional “session” endpoint ([https://server\\_name/api/1/session](https://server_name/api/1/session), see subsection 2.8) to allow

---

<sup>33</sup>Neither RFC 6749 nor RFC 6750 are violated because the authorization and token endpoint treat detached scopes like any other scope and a validation endpoint is not covered by these RFCs.

<sup>34</sup>Also, extending the authorization endpoint by accepting a “prompt” parameter as done by OpenID Connect is not feasible for user-registered clients because non-logged-in users could be redirected to malicious clients registered by other users, making the system susceptible to open redirector phishing attacks. See subsection 5.5

web applications to gather information about the current login status of a user without actually triggering any (interactive) login or permission grant procedure. This endpoint is directly accessed by the user's web browser through an XMLHttpRequest (XHR) call while setting the "withCredentials" option of the XMLHttpRequest object to true.

The call does not need any parameters and should not have any additional request headers set<sup>35</sup>. It returns a JSON object with the "member\_id" attribute set to the ID of the current user (or to null if there is no logged-in user or if a user-registered client is not authorized to obtain the login status). Since the request is done by the user's web browser, the answer is *not* authoritative for the UWUM client and must only be used as a hint. **A returned user ID MUST still be confirmed via the regular OAuth 2.0 procedure using the authorization endpoint!** In this case, the authorization endpoint will not show a login window (because the user is already logged in).<sup>36</sup>

## 2.11 Caching the login state

A successful user authentication could be cached in the session store of the UWUM client (usually at the web server side in conjunction with a cookie). This, however, can create confusion for the user because he or she might show up as being logged into the system after having logged out or vice versa. A possible solution is to use the "session" endpoint as discussed in the previous subsection 2.10 through a JavaScript which then notifies the server side of the UWUM client by redirecting the web browser if a reconfirmation of the user's login status is necessary.

In either case, UWUM clients should reconfirm that the user has not logged out at least immediately before any state changing request (e.g. posting, rating, voting, etc.) by using the validation endpoint (see subsection 2.8). This check cannot be done directly by the web browser due to security reasons (as also explained in the previous subsection 2.10).

---

<sup>35</sup>Not setting additional request headers avoids CORS pre-flight requests, see <https://www.w3.org/TR/2014/REC-cors-20140116/#cross-origin-request-with-preflight-0>

<sup>36</sup>There is a chance for a race-condition if the user simultaneously logs out. This could be solved by returning an authorization code through a CORS call. However, implementation of such a protocol is out of scope for WeGovNow and would require further security analysis.

## 2.12 Logout

### 2.12.1 Checking for logout

As explained in subsection 2.9, an access or refresh token is automatically invalidated on logout if only non-detached scopes have been requested. For all other cases, the “logged\_in” boolean field returned by the validation endpoint (see subsection 2.8) may be used to detect a logout by the user.

The “session” endpoint (as further explained in subsection 2.10) may also be used to check whether a user might have logged out (without consuming much resources on the server-side of the UWUM client).<sup>37</sup> Note, however, that a request from the web browser to the session endpoint is not suitable for the UWUM client application to validate that a user is really logged in or to securely confirm that his or her session has really ended (see subsection 2.10).

### 2.12.2 Performing logout

Depending on design criteria, logout could be performed either

- through a direct link in the UWUM navigation bar or
- through a link in the UWUM navigation bar which leads to a user page where there is a second link for the actual logout procedure.

Technical implementation requirements differ for these two cases. In the first case, the logout is performed in the context of any UWUM client; while in the second case, the final logout link or button can be displayed in the context of a web page returned by the UWUM server (which is a different origin). Due to protection against cross-site-request-forgery (CSRF), an appropriate access token or dedicated logout token would need to be part of the link in the first case (the case of using a direct link for logout). In this case, an appropriate OAuth 2.0 access token scope would need to be added to avoid unwanted exposure of the logout token (or an access token with respective scope).

A decision on this issue has not been taken yet; user interface design considerations and technical security considerations should determine which of the discussed two approaches is more suitable. Also refer to subsection 5.10, which discusses certain design limitations due to privilege separation.

---

<sup>37</sup>A future extension of UWUM could also allow UWUM clients (or their JavaScript components at the web browser side) to issue a request which is held open by the UWUM server for a set amount of time in order to allow pushing a change of the user’s login status just-in-time (see also subsection 5.4).

## 2.13 Requesting several access token scopes at once

To avoid unnecessary delays, a client may (as an extension to RFC 6749) request several access token scopes<sup>7</sup> (i.e. sets of access ranges) at once by using the parameters “scope1”, “scope2”, etc. in the Authorization Request. The corresponding result parameter “access\_token” will have “1”, “2”, etc. appended to its name (e.g. “access\_token1” etc.). Note that counting must start with “1”. It is, however, allowed to include an optional non-numbered “scope” parameter in addition to “scope1”, “scope2”, etc. The result parameters “token\_type” and “expires\_in” are never numbered or duplicated due to size limitations in the Implicit flow (maximum URL length) but always relate to all returned access tokens.

The described behavior of this subsection is not part of OAuth 2.0. Using this extension is entirely optional for the client.

## 2.14 Downgrading access token scopes

As an extension to RFC 6749, the token endpoint has been extended in such a way that it can be used to downgrade access token scopes. This feature is important for meta-APIs because according to RFC 6749, the only way to obtain a new access token without the user’s web browser is to provide a refresh token to the token endpoint.<sup>38</sup> Refresh tokens, however, are bound to a particular client and must not be shared by the client with any other party but the authorization server.<sup>39</sup>

A meta-API might receive an access token with a broader scope<sup>7</sup> than the scope necessary for calls made by the meta-API provider to another resource server. Using a greater scope than necessary for calls to resource servers, however, weakens the overall security of the system. In order to allow meta-API providers to downgrade the scope prior to using the access token, the token endpoint<sup>27</sup> accepts the string “access\_token” as value for the “grant\_type” parameter, which will tell the UWUM server that an access token (and not an authorization code or refresh token) is being presented to receive a new access token with a downgraded scope. The access token has to be provided according to the rules stated in section 2 of RFC 6750,<sup>30</sup> and one or more scopes must be requested through the “scope”, “scope1”, etc. parameters (see subsection 2.13 for details on requesting several scopes at once). Client authentication is not required. The old access token with the broader scope will not be invalidated and may still be used in future requests (e.g. to receive another access token with a different scope).

<sup>38</sup><https://tools.ietf.org/html/rfc6749#section-6>

<sup>39</sup><https://tools.ietf.org/html/rfc6749#section-10.4>

For security reasons, downgrading an access token scope will never extend the token lifetime, i.e. the returned access token will have the same remaining maximum lifetime than the access token presented to the token endpoint.<sup>40</sup>

## 2.15 Additional measures to prevent refresh token abuse

Conforming with section 10.4 of RFC 6749,<sup>41</sup> the UWUM server (LiquidFeedback) ensures that refresh tokens are bound to the client they have been issued to. As also suggested in subsection 10.4 of RFC 6749, further means to restrict refresh token abuse are implemented. Refresh tokens are replaced periodically and using a refresh token invalidates the corresponding scope<sup>7</sup> of all other previously issued refresh tokens, with the exception that refresh tokens which are still bound to a logged in user are unaffected.<sup>42</sup> An additional grace period avoids problems due to race conditions or aborted connections. This approach is similar to the example given in subsection 10.4 of RFC 6749 while being resistant against accidental race-conditions or connection aborts and allowing for a more flexible usage (e.g. different subsystems of the same client may store different refresh tokens independently).

## 2.16 Required CORS support for resource servers

Because RFC 6750 requires bearer tokens<sup>2</sup> to be accepted through the HTTP header “Authorization”,<sup>43</sup> and because the “Authorization” header is not in the list of “simple response headers” as defined by the W3C recommendation on cross-origin resource sharing,<sup>44</sup> it is inevitable for all resource servers to support cross-origin resource sharing (CORS) with the respective “Access-Control-Allow-Headers” option<sup>45</sup> set to be able to fulfill the requirements of RFC 6750. Every UWUM component acting as a resource server should therefore enable and configure CORS accordingly. See <https://www.w3.org/TR/cors/> for details.

---

<sup>40</sup>This is the reason why client authentication would not grant any extra security here and thusly can be omitted.

<sup>41</sup><https://tools.ietf.org/html/rfc6749#section-10.4>

<sup>42</sup>This is implemented by downgrading “\_detached” scopes to their corresponding non-detached scopes.

<sup>43</sup><https://tools.ietf.org/html/rfc6750#section-2.1>

<sup>44</sup><https://www.w3.org/TR/cors/#terminology>

<sup>45</sup><https://www.w3.org/TR/cors/#access-control-allow-headers-response-header>

## 2.17 HSTS

We recommend to use HTTP Strict Transport Security (HSTS)<sup>46</sup> for all WeGovNow components to increase security.

## 3 Additional endpoints for integration

Beyond user authentication and authorization, three more API endpoints are being defined for backend and UI integration:

- a “navigation” endpoint to incorporate a navigation bar,
- a “style” endpoint to retrieve style information, and
- a “client” endpoint for application and service discovery.

Prototypes for the navigation and style endpoint have been implemented; the client endpoint for application and service discovery is currently only a stub.

### 3.1 Navigation endpoint

In order to integrate all WeGovNow applications in such a way that they look and feel like a single application, all WeGovNow applications share a common navigation bar. The “navigation” endpoint of the UWUM server returns this navigation bar to be included by each WeGovNow application. This way, modifications to the navigation bar can be made at a central place without the need to change every single application.

Either a login button or the user name with a link to a user page (where logout is possible) is included in the navigation bar, depending on whether an access token is provided when calling the endpoint.<sup>47</sup> For the login button, an alternative URL may be provided by the caller of the navigation endpoint. This login URL may either be the authorization endpoint of the UWUM server with an appropriate “state” HTTP GET parameter included (note that the value must be percent-encoded<sup>48</sup>) or an URL provided by the UWUM client which initiates the OAuth 2.0 authorization and authentication procedure as described in section 2 of this document. Alternatively a unique placeholder (e.g. a GUID)

<sup>46</sup><https://tools.ietf.org/html/rfc6797>

<sup>47</sup>Also a dynamic popup menu is thinkable. However, issues with JavaScript and privilege separation in case of animated submenus according to Material Design require further consideration. Refer to subsection 5.10 in that matter.

<sup>48</sup><https://tools.ietf.org/html/rfc3986#section-2.1>

can be passed as login URL to allow caching of the rendered navigation bar and replacing the login URL locally at the UWUM client.<sup>49</sup>

Whether a structured JSON document or a pre-rendered HTML snippet is returned can be selected by another parameter passed to the navigation endpoint. A pre-rendered HTML snippet may be either returned encapsulated in a JSON response or raw for usage with the HTML5 include tag.

When the “client\_id” parameter is provided to the navigation endpoint, the corresponding client tab gets highlighted (or marked as active in case of the structured JSON document response).

It is planned to collapse the navigation bar on small screens. This feature might interfere with application specific menus; refer to subsection 5.12 for that matter.

### 3.2 Style endpoint

The style endpoint provides basic color definitions for a primary and an accent color as 8-bit RGB triplet to be able to customize the unified visual look of all WeGovNow applications for a particular installation by central configuration. Additional colors can be derived from these two base colors. If the UWUM server gets configured with colors from the Material Design color palette, the corresponding Material Design color name of the primary and the accent color is also provided.

### 3.3 Endpoint for application and service discovery

The endpoint “client” is supposed to return a list of all system applications and, if an access token is provided, a list of all registered dynamic clients for the corresponding user. Implementation of this endpoint will require storing the base URL of all system applications at the UWUM server.

Further discussion with OntoMap is required for specification and implementation of this endpoint.

## 4 Test platform

A test platform has been created in mid September to start integration with the other consortium partners.

---

<sup>49</sup>Note that the characters “<”, “>”, “&”, as well as the quotation mark character should be avoided in a placeholder string because these characters would get HTML entity encoded as described in subsection 8.1.4 of the HTML5 standard, see: <https://www.w3.org/TR/html5/syntax.html#character-references>

## 4.1 Benchmarks

The following benchmarks for integration have been defined, whose fulfillments have been published in the weekly status reports.

**Client URLs established** A client application (resource server and/or relying party) has been installed and its base URL and redirection endpoint<sup>50</sup> has been communicated to the consortium.

**SSL key and certificate for end-users** A private key and a publicly trusted SSL certificate has been created for the end-user web interface and SSL connections to that interface have been successfully tested.

**Certificate signing request (CSR) for UWUM API** A private key for accessing the UWUM API and a corresponding certificate signing request (CSR) has been created and submitted to FlexiGuided GmbH (LiquidFeedback).

**SSL certificate for UWUM API** A signed certificate for the UWUM API client key has been sent back to the consortium member and their client application has successfully established a secured connection with the UWUM server.

**Authorization endpoint accessed** The client application can redirect an end-user to the UWUM authorization endpoint.<sup>51</sup>

**Authorization endpoint error response handling** The client application is capable of receiving authorization errors<sup>52</sup> through its redirection endpoint<sup>50</sup> and displaying it to the end-user.

**Access token request (including end-user identification)** The client application has successfully received an authorization code and identified the end-user through an access token request.<sup>53</sup>

**Access token request error handling** The client application is capable of properly processing errors during the access token request.<sup>54</sup>

**Using access tokens for API calls to other components** The client application has successfully used an access token to perform a LiquidFeedback API call.

---

<sup>50</sup><https://tools.ietf.org/html/rfc6749#section-3.1.2>

<sup>51</sup><https://tools.ietf.org/html/rfc6749#section-4.1.1>

<sup>52</sup><https://tools.ietf.org/html/rfc6749#section-4.1.2.1>

<sup>53</sup><https://tools.ietf.org/html/rfc6749#section-4.1.3>

<sup>54</sup><https://tools.ietf.org/html/rfc6749#section-5.2>

**Access token verification** The client application is capable of verifying the validity and scope of an access token.

**Accepting access tokens from other components** The client application provides at least one API call where an access token is used for authorization.

**Accepting access tokens as “Authorization” header** In conformance with RFC 6750 (Bearer Token Usage), the client application (resource server) accepts access tokens through the authorization request header field.<sup>55</sup>

**Cross-origin resource sharing** The client application allows cross-origin resource sharing (CORS) as described in subsection 2.16 of this document.

**Cross-application navigation** The UWUM navigation bar has been successfully integrated into the client application.

**IPv6** IPv6 capabilities have been tested.

## 5 Technical challenges

In this section, we will describe obstacles encountered during implementation and during integration with the consortium partners as well as respective solutions.

### 5.1 Third party clients (non-registered clients vs. dynamic registration)

OAuth 2.0 demands client registration but does not specify how such client registration is to be implemented.

“Before initiating the protocol, the client registers with the authorization server. The means through which the client registers with the authorization server are beyond the scope of this specification but typically involve end-user interaction with an HTML registration form.”<sup>56</sup>

Manual client registration, however, is only suitable for a service-centered approach where a software provides only a single service (e.g. Facebook, Google, Twitter, etc). An open source solution, however, could be installed at several sites by different service providers. It is therefore not sufficient to register a client

<sup>55</sup><https://tools.ietf.org/html/rfc6750#section-2.1>

<sup>56</sup>Ed. D. Hardt: The OAuth 2.0 Authorization Framework, October 2012. Section 2 (Client Registration), <https://tools.ietf.org/html/rfc6749#section-2>

at a single service provider if this client shall be usable for any service provider using the UWUM server software.

One possible solution would be the creation of a central (i.e. world-wide) UWUM client registry. Such central client registry, however, could be a single point of failure and would empower a central authority to control usage of the UWUM protocol (e.g. it would be possible to block certain clients). We consider this approach contrary to the concepts of open source and open data.

Therefore, we implemented a dynamic client registration protocol that keeps implementational complexity at a minimum while providing good security properties which outperforms many other solutions for client registration due to requiring direct access to the DNS zone of the domain (for adding a TXT record) or credentials (a publicly trusted TLS certificate with corresponding key) that should be accessible only by the domain owner. Dynamic client registration is described in subsection 2.4.2 of this document.

## 5.2 TLS client side certificates and web browser behavior

Web server software often offers three different settings for handling TLS client certificates:

- client-side certificates disabled,
- optional client-side certificate,
- mandatory client-side certificate.

Often these settings can be made only on a per-domain basis (i.e. for each virtual host). Furthermore, enabling client-side certificates (even if set to “optional”) will cause web browsers to show up a dialoge when accessing pages on that domain.

For these reasons, a separate hostname has to be used for API endpoints when a TLS client-side certificate is to be provided (which affects the token endpoint<sup>27</sup>). The UWUM server will have to provide a configuration endpoint where dynamic clients may retrieve a deviant domain for the token endpoint; and dynamic UWUM clients (see subsection 2.4.2) will have to query this configuration endpoint prior to using the token endpoint).

## 5.3 Multi-domain certificates

TLS certificates may be issued for more than one domain using the “Subject Alternative Name” (SAN) extension. The current implementation of Liquid-Feedback, however, relies on an HTTP reverse proxy to include the distinguished

name (DN)<sup>57</sup> of the certificate in a designated HTTP header. Some reverse proxy software, namely “NGINX” which is recommended for use with LiquidFeedback, does not properly support transmitting a domain list from the SAN extension. In case of “NGINX”, header line folding<sup>58</sup> is used to pass multiple domain names from a TLS certificate to the respective backend (e.g. LiquidFeedback). Header line folding, however, has recently been deprecated by RFC 7230,<sup>59</sup> and it is not supported by LiquidFeedback (and not even by “NGINX” for incoming requests). The problem of header line folding in the context of multi-domain TLS certificates has also been discussed in the “NGINX” issue tracker under ticket #857.<sup>60</sup> The issue is currently not classified as bug<sup>61</sup> and it is unclear when a patch will be incorporated into the software.

For the technical difficulties explained above, we refrained from supporting multi-domain certificates at this stage. In case of UWUM clients approved by the municipality or operator of LiquidFeedback, this shouldn’t be a problem anyway because the certificate authority will be under the control of the operator, such that it is easy to create a certificate using the DN/CN property. For dynamically registered clients, an alternative mechanism using DNS TXT records is available (see subsection 2.4.2).

If multi-domain certificates are supported in the future, it is vital that the token endpoint requires the “client\_id” parameter to be set for all clients authenticating with such a multi-domain certificate. This way, code substitution attacks<sup>23</sup> can be repelled. (Note that RFC 6749 requires the “client\_id” parameter to be set only if the client is not authenticating with the authorization server,<sup>22</sup> but this does not work for multi-domain certificates.)

## 5.4 Outdated logins

While a successful OAuth 2.0 authorization procedure (using the Authorization Code flow<sup>1</sup>) can be used to confirm that a user is logged in at the particular time of the Access Token Response<sup>24</sup>, an UWUM client obviously can’t assume that the login will be still valid at any later time.

UWUM currently provides two methods to check if a user has logged out; these are explained in subsection 2.12.1 of this work report. Considerations in regard to purposeful caching of the user’s login status are found in subsection 2.11.

Even if no caching of the login status is performed, there is still the possibility that a user opens WeGovNow with two different browser windows or browser tabs.

---

<sup>57</sup>The DN contains a single domain as CN (common name).

<sup>58</sup><https://tools.ietf.org/html/rfc2616#section-2.2>

<sup>59</sup><https://tools.ietf.org/html/rfc7230#appendix-A.2>

<sup>60</sup><https://trac.nginx.org/nginx/ticket/857>

<sup>61</sup><https://trac.nginx.org/nginx/ticket/857#comment:2>

He or she might then log out in one window and afterwards switch to the other window where the logout has not been noticed yet, which creates confusion for the user. A possible solution is to regularly check if the user has logged out by utilizing the cross-origin-resource-sharing (CORS) XML-HttpRequest (XHR) as explained in subsection 2.10.

Regular requests to detect logouts, however, cause unnecessary resource consumption for all involved components. A better approach would be to have a permanent TCP connection between the web browser and the UWUM server (or alternatively between the UWUM client and the UWUM server if at the same time there is a permanent connection between the web browser and the UWUM client). There are different technologies thinkable for this approach. One method is to keep an XML-HttpRequest open for a set amount of time during which the server is capable of sending a message directly to the web browser. (The request has to be repeated after timeout or after a message has been received, whichever happens first.) Another technique would be to use WebSockets. None of these additional techniques have been implemented yet.

## **5.5 Susceptibility to open redirector phishing attacks when allowing login checks through web browser redirection**

Subsection 2.10 mentioned that an interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechanism on its own, and our research concluded that any form of redirection-based mechanism for providing this functionality<sup>62</sup> would be susceptible to open redirector phishing attacks as described in subsection 4.2.4 of RFC 6819 (“OAuth 2.0 Threat Model and Security Considerations”)<sup>63</sup> as long as third parties are capable of registering a malicious client with a corresponding redirection URI<sup>12</sup> that is under the control of the third party.

The previously mentioned subsection of the threat model and security considerations document (RFC 6819) suggests client registration with redirect URI registration (and avoiding redirects to any non-registered redirect URI)<sup>64</sup> as only countermeasure for this threat. However, this countermeasure only works when manual client registration (and manual approval through the operator of the UWUM server) is mandatory. It particularly fails if dynamic client registration (e.g. as described in subsections 2.4.2 and 5.1 of this work report) is allowed.

---

<sup>62</sup>e.g. accepting a “prompt” parameter as done by OpenID Connect, see [http://openid.net/specs/openid-connect-core-1\\_0.html#AuthRequest](http://openid.net/specs/openid-connect-core-1_0.html#AuthRequest)

<sup>63</sup><https://tools.ietf.org/html/rfc6819#section-4.2.4>

<sup>64</sup><https://tools.ietf.org/html/rfc6819#section-5.2.3.5>

Luckily, the technique of cross-origin resource sharing (CORS) allowed for the development of an alternative to the redirect-based approach. Subsection 2.10 explains the mechanism.

## 5.6 Handling of updated user related data (e.g. user's e-mail addresses)

When a WeGovNow application wants to send notification e-mails to users, it is not adequate to retrieve the e-mail address only once from UWUM as the notification e-mail can be changed by the user at any time. Such a change needs to be reflected by all applications using this e-mail address. Therefore an application needs to retrieve the current notification e-mail address *directly* before using it, in fact again before every usage.

For that purpose, another API endpoint `/api/1/notify_email` (GET) can be used (using an access token with the “notify\_email” scope). To be able to retrieve the e-mail address while the user is not currently logged in, it will be necessary to request the “notify\_email\_detached” scope when identifying the user and to store the received refresh token permanently. The suffix “\_detached” requests a scope for detached usage, i.e. for usage even after the user logs out.<sup>65</sup>

Similar situations can occur related to other member properties stored in one application but used in another one, e.g. the screen name. But these seem not to be as critical as to avoid using an outdated e-mail address. Such properties could be cached for a limited time before retrieving them again from the application storing this property.

## 5.7 Race conditions with refresh token rotation

As suggested in subsection 10.4 of RFC 6749,<sup>41</sup> refresh token rotation is employed to provide better security properties (e.g. in case of exposed refresh tokens and client certificates, or in case of the existence of a single compromised certificate authority which would render authentication of dynamic clients insecure).

Unfortunately, RFC 6749 does not specify how old refresh tokens are invalidated. Section 6 of RFC 6749 only says that<sup>66</sup>

- the authorization server MAY issue a new refresh token, in which case
- the client MUST discard the old refresh token and replace it with the new refresh token, and

<sup>65</sup>Note that when exchanging a refresh token for an access token after the user has been logged out, an UWUM client must also explicitly request the “\*\_detached” scope(s) it needs, e.g. “notify\_email\_detached” using the scope parameter of the `/api/1/token` endpoint.

<sup>66</sup><https://tools.ietf.org/html/rfc6749#section-6>

- the authorization server MAY revoke the old refresh token.

Always revoking the old refresh token after transmission can have a bad effect on system stability, considering that responses might be interrupted. Furthermore, multiple backends of an UWUM client could simultaneously access the token endpoint. Such legit accesses by two legit backends of the same client would need to be distinguished from accesses by a legit client and a malicious third party who obtained a copy of a refresh token.

Subsection 2.15 explains the mechanisms employed by the UWUM server to mitigate the risk of refresh token abuse while solving the problems stated above.

## 5.8 Creating a set of suitable access token scopes

A useful set of access token scopes<sup>7</sup> is a vital aspect of privilege separation. From a security point of view, scopes should be as fine-graded as possible, particularly there should be different scopes for different applications (e.g. an application that wishes to rate user contributions in application X does not need an access token that allows to rate user contributions in application Y). Extensibility, on the other hand, would be complicated if access token scopes always refer to a single application (i.e. a single resource server in this context). Furthermore, it is a goal that the WeGovNow platform looks and feels like a single integrated application. When users grant access scopes to third party clients, such application-based scopes would be difficult to understand for the user, which by itself can have bad influence on the overall system security.

We therefore decided to provide a set of generic access token scopes as listed in subsection 2.6. For future extensions, see footnote 20.

## 5.9 Misconceptions regarding scopes vs. user privileges

Scopes must not be mistaken for user privileges. I.e. a scope does *not* grant a privilege to a user; it just means an application can trigger an action within the scope *if* the user is authorized to perform the action. For example, an application needs the scope “vote” to cast a vote on behalf of the user but casting a vote will only work if the user has the necessary voting privileges.

Programmers of UWUM clients must keep these differences in mind and execute an action only if both the scope and the users privileges are sufficient for the respective action.

## 5.10 JavaScript integration and privilege separation

Dynamically sharing JavaScript code between UWUM clients or between the UWUM server and an UWUM client violates privilege separation because it would enable one component to execute code in the security context of another origin. For example, one application 'A' could send a harmful JavaScript to be included in a web page returned by another application 'B' which then discloses the session cookie for application 'B' to application 'A'.

For this reason, the common navigation bar as returned by the navigation endpoint (see subsection 3.1) currently does not include any JavaScript code. UWUM clients may therefore even consider to sanitize the returned HTML code in such a way that any JavaScript is removed or rejected.

Interface design decisions, however, might suggest to use JavaScript for the navigation bar. Material design, for example, requires popup-menus to be animated, which cannot be done with CSS alone. Another reason for JavaScript might be dynamic modifications of the navigation bar (e.g. collapsing the navigation bar to a menu icon) depending on the screen size or the device of the user. Also other integration techniques might suggest the use of JavaScript.

An alternative to dynamically provided JavaScripts by the UWUM server would be a common library to be included locally by each WeGovNow component. Whenever this library is updated, administrators of each component can look over it before incorporating it. While this approach provides proper privilege separation, its downside would be the administrative overhead.

At least in regard to the navigation bar, it would eventually need to be decided whether

- there will be no JavaScript used by the navigation bar,
- the UWUM server will dynamically return JavaScript code for the navigation bar, or
- each WeGovNow component needs to include a pre-distributed JavaScript.

## 5.11 Logout through navigation bar

The common WeGovNow navigation bar (as returned by the "navigation" endpoint, see subsection 3.1) should also include a possibility to logout. Due to protection against cross-site-request-forgery (CSRF) and because the navigation bar will be included in responses from different web servers (different "origins"), a simple logout link does not work. Subsection 2.12.2 deals with different approaches to this problem.

## 5.12 Collapsing navigation bar and application menu

In case of mobile devices, it may be desirable to collapse the navigation bar to a single menu icon displayed in the corner of the screen. Despite the technical problems in regard to JavaScript (which are discussed in subsection 5.10), there is also a challenge in regard to a potentially existent second menu bar which is provided by the particular application currently selected.

It could be difficult for the user if two menu icons are being displayed (i.e. a meta-menu, which covers the entries of the navigation bar, and an application specific menu). A potential solution could be to combine both menus into a single one. In this case, however, the considerations of subsection 5.10 still apply.

## 5.13 UWUM clients without user interface

In addition to UWUM clients having a user interface, there are also WeGovNow applications thinkable which do not have any (end-)user interface. This includes both meta-API providers as well as other service components. In the context of WeGovNow, one meta-API provider could be OntoMap.

The current UWUM specification enables the development of meta-APIs because access tokens are not bound to a particular UWUM client and can be downgraded in regard to their access token scope (of which the latter is important for security, see subsection 2.14). Thus, a meta-API can simply require its callers to provide a valid access token which then can either be used directly or downgraded for further requests performed by the meta-API provider to other resource servers.

Nonetheless, the mechanisms described in this work report still require privileges that are bound to a particular user. For UWUM clients requiring access privileges that are not tied to a particular user (e.g. clients which aggregate data of all users and publish that information), the Client Credentials Grant<sup>67</sup> should be implemented.

## 5.14 Client authentication for resource servers

While UWUM enables (a) its clients to authenticate users and (b) resource servers to verify user authorization (both explained in section 2), it does not enable resource servers to authenticate clients. Such client authentication might be required by applications that want to establish a trusted channel to another application independently of user authorization.<sup>68</sup>

<sup>67</sup><https://tools.ietf.org/html/rfc6749#section-4.4>

<sup>68</sup>An example could be OntoMap logging actions executed at other applications (which are then reported to OntoMap by the respective application with client authentication enabled).

Unfortunately, neither OAuth 2.0 nor UWUM enable applications to verify the identity of another application. Even if OAuth 2.0 uses client authentication for a variety of reasons<sup>69</sup> for the authorization endpoint<sup>26</sup>, it doesn't provide such an authentication method to other applications. Extending the OAuth 2.0 work flow in this matter (e.g. by returning the `client_id` when an access token is presented to the validation endpoint<sup>70</sup>) would rise some issues:

- Tying an access token (a bearer token<sup>2</sup> in case of UWUM) to a particular client does not make sense in case of applications that behave both as an OAuth 2.0 resource server and as a client (e.g. meta-API providers or applications which provide an API and have to perform further API calls to complete a requested action). Also, client impersonation would be possible. To give an example: if a received access token is tied to a particular client A, and if application A uses this access token to perform an action at application B, then application B would be able to impersonate application A. Furthermore, application B couldn't use the access token to authenticate as application B when performing further requests at the API of another application C.
- Using a custom scope to identify the origin of a request (e.g. a scope "I\_am\_appX") would also enable client impersonation (e.g. any application who receives an access token with the scope "I\_am\_appX" could then impersonate application X). An alternative could be to use scopes that reflect better the particular action to be performed, e.g. a scope "write\_appXs\_log\_at\_appY". It is self-evident that this would increase the number of scopes drastically (possibly quadratically), which, in turn, might create a maintenance/configuration mess. Other than that, there is another problem with using scopes for client authentication: following RFC 6750, there can only be one bearer token per request.<sup>2</sup> If a client needs to use a received access token for an API call at another component, then this access token could not be used to authenticate that client because it won't have the necessary scope. One possible solution could be to allow adding scopes to an existing access token or extend RFC 6750 in such a way that multiple access tokens could be used per request. All those solutions, however, go far beyond OAuth 2.0 and would require extra implementation work for all consortium partners. In the end, the created solution wouldn't be OAuth 2.0 anymore.

<sup>69</sup>See beginning of subsection 2.4.2 of this report.

<sup>70</sup>See subsection 2.8 for an explanation of the validation endpoint.

The straight-forward way of authenticating clients is to use the existing mechanism already employed by all UWUM clients: TLS client-side certificates. This, however, requires TLS client certificate checking by each resource server that needs to authenticate other clients.

© 2016 FlexiGuided GmbH, Berlin

## **Appendix 2.4**

### Navigation Bar

## Appendix 2.4 - Navigation Bar

The WeGovNow navigation bar can be retrieved as HTML+CSS+JS or as raw data in JSON format. In the latter case, it should be dynamically rendered with material design like style, in each of the core components side..

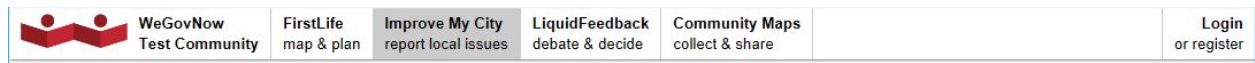


Figure 1. The landing page with a highlighted component.

The component should highlight the corresponding tab to facilitate the user in the navigation, see Figure 1. The selected item is flagged by UWUM for convenience.

### Responsive layouts

The navigation bar is mobile friendly, meaning that it is adjustable to various screen widths.



Figure 2. The mobile friendly version

Figure 2 depicts the navigation bar collapsed.



Figure 3. The expanded mobile version

The navigation bar is composed of:

- 1) The markup: It is just a <header> HTML tag

- 2) Inline CSS styling
- 3) Inline javascript (absolute bare minimum standard javascript without external libraries)
- 4) The WeGovNow logo in PNG format

## **Appendix 3.3**

### **Work report on FirstLife API**

# FirstLife API reference

(v5.0)

2017-01-30

Department of Computer Science  
University of Turin, Italy

## [Introduction](#)

## [Authentication](#)

[Obtaining the access token](#)

[Refreshing the access token](#)

## [FirstLife entities](#)

[Categories and Entities](#)

[Entity types](#)

[Places](#)

[Events](#)

[Groups](#)

[News](#)

[Articles](#)

[Working with entities](#)

[Create entities](#)

[Update entities](#)

[Delete an entity](#)

[Retrieve and entity](#)

[Retrieve related entities](#)

[Events located in a place](#)

[Articles associated to an entity](#)

[Sub-entities](#)

[News associated to an entity](#)

[Sub-groups](#)

## [Adding descriptions, comments and images](#)

[Working with descriptions](#)

[Retrieve all entity descriptions](#)

[Adding a description](#)

[Update a description](#)

[Delete a description](#)

[Working with comments](#)

[Retrieve all entity comments](#)

[Adding a comment](#)

[Update a comment](#)

[Delete a comment](#)

[Working with images](#)

[Retrieve all entity images](#)

[Adding an image](#)

[Update an image](#)

[Delete an image](#)

## Introduction

This documents contains details on FirstLife API that allows the user to create and use entities defined in the data model.

## Authentication

Currently, FirstLife backend supports OAuth 2 authorization through Authorization Code Flow Pattern, in which FirstLife backend plays the role of Resource Server.

Actually, It supports two authorization services:

- UWUM, provided by WeGovNow core platform
- FIRSTLIFE, provided by Unito

For both services, the authorization data must be specified in the request headers, at each operation call. In details

- access token
- authorization server

For example:

```
"Authorization" = "Bearer uPLf6lOjua2P4CmA"  
"Authentication_server" = "FIRSTLIFE"
```

The list of authentication servers values supported by FirstLife API is

- "FIRSTLIFE" for the FirstLife authentication server (OAuth 2.0)
- "UWUM" for the Unified WeGovNow Users Management system (OAuth 2.0)

## Obtaining the access token

The FirstLife API offers a custom operation, for each authentication server, to get a valid access token starting from the authorization code:

For FirstLife server:

POST /v5/fl/tokens/fl\_auth

payload (example)

```
{  
  "code": "pPbNr0sG2JqjicsF"
```

```
}
```

For UWUM server:

POST /v5/fl/tokens/fuwum\_auth

payload

```
{  
  "code": "pPbNr0sG2JqjicsF"  
}
```

In the following, an example of the returned access token info:

```
{  
  "token": {  
    "auth_server": "UWUM",  
    "member_id": 125,  
    "access_token": "uPLf6lOjua2P4CmA",  
    "refresh_token": "ZGQufnRqVQO6WfAj",  
    "expires_in": 3600,  
    "token_type": "bearer",  
    "created": "2017-01-10T13:14:52.460Z",  
    "expiration": "2017-01-10T14:14:52.336Z",  
    "scope": "identification read_authors read_contents  
read_identities read_ratings",  
    "id": "5874de4ca05793ad7b4d9609",  
    "member": {  
      "id": 125,  
      "identification": "UNITO/FirstLife Test Account 07",  
      "name": "Jonh Doe"  
    }  
  }  
}
```

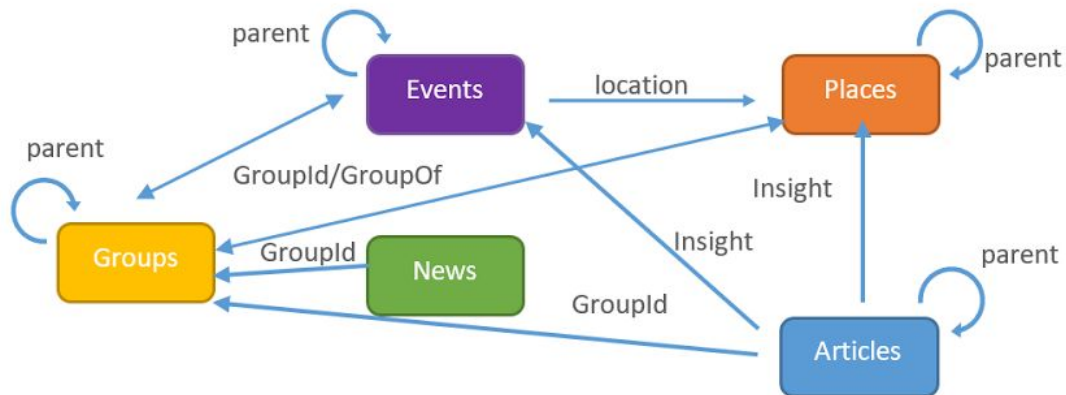
## Refreshing the access token

At each operation call, the FirstLife API checks the access token validity and expiration and, if necessary, it provides to refresh the token through the use of the previously stored refresh token information. In this case, the new access token is included in the headers response, with the key “new\_access\_token”.

## FirstLife entities

Currently, the FirstLife data model allows the definition of different kind of entities, like places, events, news and articles. All entity types share a common set of properties, like geographical information, ownership, name, categorization, etc. At the same type, each entity type is defined by a set of specific attributes: for examples, events can be described by the starting time and the duration. Moreover, entities can be linked by relations: for example an event is linked to a place by using the relation “location”, or a sub-place is linked to its container by a “parent” relationship. Finally, FirstLife allows the definition of Groups. They are meant to collect all other entities, in order to provide a representation of a

group activities and create a collective responsibility. Groups can also be included in other entities allowing patterns such as group of participants of an event.



All entity shares a GeoJson structure (<http://geojson.org/>), with a set of common properties. The following table summarizes the top level structure of a FirstLife entity:

Property	Type	Notes
id	String	id of the entity (auto generated, immutable)
type	String	constant value: "Feature"
properties	Object	GeoJson list of properties
geometry	Object	GeoJson geometry object

At this moment, FirstLife supports Point as geometry type for entities, specified according to the GeoJson specification in the geometry property.

The list of properties shared by all entities is listed in the following table:

Property	Type	Mand atory	Notes
id	String	Yes	id of the entity (auto generated)
entity_type	String	Yes	supported values: "FL_PLACES", "FL_EVENTS", "FL_GROUPS", "FL_ARTICLES", "FL_NEWS"
domain_id	Number	Yes	id of the domain to which the entity belongs
name	String	Yes	Name of the entity
parent_id	String	No	id of the parent entity
tags	Array of String	No	tags associated to the entity

owner	String	Yes	id of the user who created the entity (auto generated starting from authorization token, immutable)
owner_display_name	String	Yes	Label of the user who created the entity (auto generated from the authorization token, immutable)
updater	String	Yes	id of the last user who modified the entity (auto generated from the authorization token, immutable by the user)
updater_display_name	String	Yes	Label of the last user who modified the entity (auto generated from the authorization token, immutable by the user)
timestamp_insert	Date	Yes	Auto generated, immutable
last_update	Date	Yes	Auto generated, immutable by the user
categories	Object	Yes	
thumbnail	String	No	url to thumbnail image
valid_from	Date	No	Validity constraint of the entity
valid_to	Date	No	Validity constraint of the entity
group_id	Number	No	id of the group to which the entity belongs
semantic_url	String	No	
self	String	Yes	Deep link to the entity (auto generated, immutable)

## Categories and Entities

In FirstLife, each entity is associated to a multidimensional category system, enabling a way to classify Places, Events, etc, into a multi-facet domain system.

List of categories describing a specific entity facet is represented by the definition of a Category Space. In each FirstLife domain, each entity can be classified by the association with one or more Category Space. The number of categories, belonging to a Category Space, that can be associated to an instance of an entity type (in a given domain) can be decided at design time by the domain administrator.

To obtain the list of defined domains:

GET /v5/fl/domains

To obtain all details about categories defined in a specific domain:

GET /v5/fl/domains/[domain\_id]/categories

## Entity types

Within FirstLife, the user can create different type of entities:

- Places
- Events
- Groups
- News
- Articles

### Places

No additional properties specified

### Events

The additional properties are listed in the following table

Property	Type	Mandatory	Notes
location	String	Yes	id of the entity where the event take place
door time	Date	Yes	Event opening date and time
duration	Number	No	Duration of the event, in minutes

### Groups

The additional properties are listed in the following table

Property	Type	Mandatory	Notes
administrator	String	Yes	id of the administrator of the group
group_of	String	No	id of the parent group

### News

The additional properties are listed in the following table

Property	Type	Manda	Notes
----------	------	-------	-------

		tory	
news_of	String	Yes	id of the entity to which the news belongs
message	String	Yes	content of the news

## Articles

The additional properties are listed in the following table

Property	Type	Mandatory	Notes
article_of	String	Yes	id of the entity to which the article belongs
text	String	Yes	content of the article

## Working with entities

### Create entities

POST /v5/fl/Things

Payload: a GeoJson object representing the entity.

For example:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [7.2656, 45.47899]
  },
  "properties": {
    "entity_type": "FL_PLACES",
    "domain_id": 1,
    "name": "Test for places",
    "tags": ["building", "free"],
    "categories": [{
      "category_space": {
        "id": 1
      },
      "categories": [{
        "id": -6
      }]
    }, {
      "category_space": {
        "id": 13
      },
    },
  }
}
```

```

        "categories": [{
            "id": -92
        }]
    }]
}

```

It is important to notice that category spaces and categories are specified by using the corresponding id. For example:

```

"categories": [
  {
    "category_space": {
      "id": 1,
      "name": "Uso del Luogo",
      "slug": "Uso del Luogo"
    },
    "categories": [
      {
        "id": -6,
        "name": "Attività per il sociale",
        "category_index": 3,
        "icon_name": "ion-android-people"
      }
    ]
  },
  {
    "category_space": {
      "id": 13,
      "name": "Tipo di Luogo",
      "slug": "Tipo di Luogo"
    },
    "categories": [
      {
        "id": -92,
        "name": "Spazio privato",
        "category_index": 3,
        "icon_name": "ion-ios-home"
      }
    ]
  }
]

```

If all constraints are correct, the operation returns the new entity:

```

{
  "properties": {
    "name": "Test for places",
    "tags": [
      "building",
      "free"
    ],
    "entity_type": "FL_PLACES",
    "categories": [

```

```

{
  "category_space": {
    "id": 1,
    "name": "Uso del Luogo",
    "slug": "Uso del Luogo"
  },
  "categories": [
    {
      "id": -6,
      "name": "Attività per il sociale",
      "category_index": 3,
      "icon_name": "ion-android-people"
    }
  ]
},
{
  "category_space": {
    "id": 13,
    "name": "Tipo di Luogo",
    "slug": "Tipo di Luogo"
  },
  "categories": [
    {
      "id": -92,
      "name": "Spazio privato",
      "category_index": 3,
      "icon_name": "ion-ios-home"
    }
  ]
}
],
"domain_id": 1,
"owner": "57c13b1874b4c257989995ce",
"owner_display_name": "Claudio",
"updater": "57c13b1874b4c257989995ce",
"updater_display_name": "Claudio",
"timestamp_insert": "2017-01-11T16:05:08.238Z",
"last_update": "2017-01-11T16:05:08.243Z",
"id": "587657b41015d0b79cb50900",
"status_id": 1
},
"geometry": {
  "type": "Point",
  "coordinates": [
    7.2656,
    45.47899
  ]
},
"type": "Feature",
"id": "587657b41015d0b79cb50900"
}

```

## Update entities

```
PUT /v5/fl/Things/{thingId}
```

Notice that it is sufficient to indicate only properties user want to update. At the end of the update, the updated entity is returned.

Some examples.

Update name and location of a FirstLife event:

```
{
  "properties": {
    "name": "Updated name",
    "location": "015d0b79cb509045877a8eb1"
  }
}
```

Update the list of tags (to add a new tag, you have to update the entire array):

```
{
  "properties": {
    "tags": ["old_tag_1", "old_tag_2", "new_tag"]
  }
}
```

## Delete an entity

```
DELETE /v5/fl/Things/{thingId}
```

## Retrieve and entity

```
GET /v5/fl/Things/{thingId}
```

## Retrieve related entities

Entities can be linked by relations: for example an event is linked to a place by using the relation “location”, or a sub-place is linked to its container by a “parent” relationship.

### Events located in a place

```
GET /v5/fl/Things/{thingId}/located
```

Entities are linked by the `location` property of FirstLife events.

## Articles associated to an entity

GET /v5/fl/Things/{thingId}/articles

Entities are linked by the `article_of` property of FirstLife articles.

## Sub-entities

GET /v5/fl/Things/{thingId}/children

Entities are linked by the `parent_id` property of FirstLife (sub) entities

## News associated to an entity

GET /v5/fl/Things/{thingId}/news

Entities are linked by the `news_of` property of FirstLife news

## Sub-groups

GET /v5/fl/Things/{thingId}/groupsof

Sub-groups are linked by the `group_of` property

# Adding descriptions, comments and images

Entities can be enriched by adding different weak entity objects like descriptions, comments and images

## Working with descriptions

Property	Type	Mandatory	Notes
id	String	Yes	id of the description (auto generated, immutable)
title	String	Yes	title of the description
description	String	Yes	content of the description
owner	String	Yes	id of the user who created the description (auto generated starting from authorization token)
owner_display_name	String	Yes	Label of the user who created the description (auto generated from the authorization token)
updater	String	Yes	id of the last user who modified the description (auto generated from the authorization token)

updater_display_name	String	Yes	Label of the last user who modified the description (auto generated from the authorization token)
timestamp_insert	Date	Yes	Auto generated, immutable
last_update	Date	Yes	Auto generated, immutable

## Retrieve all entity descriptions

GET /v5/fl/Things/[thingId]/descriptions

## Adding a description

POST /v5/fl/Things/[thingId]/descriptions

Payload example:

```
{
  "title": "Description title",
  "description": "Description content...."
}
```

## Update a description

PUT /v5/fl/Things/[thingId]/descriptions/[descriptionId]

Payload example:

```
{
  "title": "NEW Description title"
}
```

## Delete a description

DELETE /v5/fl/Descriptions/[descriptionId]

DELETE http://localhost:3090/v5/fl/Things/[thingId]/descriptions/[descriptionId]

## Working with comments

Property	Type	Mandatory	Notes
id	String	Yes	id of the description (auto generated, immutable)

message	String	Yes	content of the description
owner	String	Yes	id of the user who created the description (auto generated starting from authorization token)
owner_display_name	String	Yes	Label of the user who created the description (auto generated from the authorization token)
updater	String	Yes	id of the last user who modified the description (auto generated from the authorization token)
updater_display_name	String	Yes	Label of the last user who modified the description (auto generated from the authorization token)
timestamp_insert	Date	Yes	Auto generated
last_update	Date	Yes	Auto generated

## Retrieve all entity comments

GET /v5/fl/Things/[thingid]/comments

## Adding a comment

POST /v5/fl/Things/[thingid]/comments

Payload example:

```
{
  "message": "Comment content...."
}
```

## Update a comment

PUT /v5/fl/Things/[thingId]/comments/[commentId]

Payload example:

```
{
  "message": "NEW comment"
}
```

## Delete a comment

DELETE /v5/fl/Comments/[commentId]

DELETE /v5/fl/Things/[thingId]/comments/[commentId]

## Working with images

Property	Type	Mandatory	Notes
id	String	Yes	id of the description (auto generated, immutable)
image_name	String	Yes	image caption
filename	String	Yes	name of the file (auto generated by the framework)
owner	String	Yes	id of the user who created the description (auto generated starting from authorization token)
owner_display_name	String	Yes	Label of the user who created the description (auto generated from the authorization token)
updater	String	Yes	id of the last user who modified the description (auto generated from the authorization token)
updater_display_name	String	Yes	Label of the last user who modified the description (auto generated from the authorization token)
timestamp_insert	Date	Yes	Auto generated
last_update	Date	Yes	Auto generated

### Retrieve all entity images

GET /v5/fl/Things/[thingid]/images

### Adding an image

POST /v5/fl/Things/[thingid]/images

Payload example:

```
{
  "image_name": "Image content....",
  "filedata": "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/ ..."
}
```

### Update an image

PUT /v5/fl/Things/[thingId]/images/[imageId]

Payload example:

```
{  
  "image_name": "NEW image comment"  
}
```

## Delete an image

DELETE /v5/fl/Images/[imageId]

DELETE /v5/fl/Things/[thingId]/images/[imageId]

## **Appendix 3.5**

### **Work report on extending the LF Core**

# Work report on extending the LiquidFeedback Core

Jan Behrens, Andreas Nitsche

June 6, 2016

© 2016 FlexiGuided GmbH, Berlin

## 1 Planned tasks

As a prerequisite for implementing a REST API for LiquidFeedback, the objective was to prepare LiquidFeedback’s backend (“LiquidFeedback Core”) in such way that its decision-making processes can be integrated into other systems, particularly extending it by adding a feature of geo-tagging user input as well as allowing geographic searches (geospatial indexing).

### 1.1 Removing “membership” in subject areas

LiquidFeedback version 3.1 (as well as version 3.2) relies on explicit member registration per subject area in order to calculate a reference population that is then used in the 4-phase decision making process.<sup>1</sup>

Registration in a subject area is infeasible when integrating LiquidFeedback with map-based frontends. Some users might not even be aware of the available subject areas in LiquidFeedback, hence the registration counts would be arbitrary, which, in turn, would influence LiquidFeedback’s decision making process in a non-predictable way.

---

<sup>1</sup>The reference population is used both for the first supporter quorum (issue quorum between admission and discussion phase) and for the second supporter quorum (initiative quorum between verification and voting phase).

Previous work has been published on a possible modification of the LiquidFeedback decision making process.<sup>2</sup> As of December 14, 2015, it was decided by the LiquidFeedback developers to generally adopt such a modification while some simplifications were proposed.<sup>3</sup> A working implementation, however, was still outstanding as of April 2016, and had thus to be created as part of WP3 in order to allow for an integration of LiquidFeedback into the other map-based components.

## 1.2 Adding necessary data structures for geo-tagging

LiquidFeedback 3.x has no means of storing geographical information. In order to allow for LiquidFeedback initiatives and suggestions to be displayed on a map, the necessary data fields needed to be added to the data structures of the LiquidFeedback backend ("LiquidFeedback Core").

## 1.3 Adding a geospatial index to allow for geographic searching and/or selection of user data

The planned REST API for LiquidFeedback needs to include search functions (e.g. boxes parametrized with latitude/longitude ranges, or radial searches with a center and a radius). Considering a possibly huge number of datasets, spatial indexing is a requirement for offering such API calls. LiquidFeedback currently does not use a database with geospatial capabilities.<sup>4</sup>

A common solution for geospatial data processing with PostgreSQL is PostGIS (see <http://postgis.net/>). Due to its high number of further dependencies<sup>5</sup> as well as its license<sup>6</sup>, however, it was necessary to consider alternatives for geospatial data processing.

---

<sup>2</sup>Jan Behrens, Andreas Nitsche, Björn Swierczek: A Finite Discourse Space for an Infinite Number Of Participants. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 4 (2015-07-28). ISSN 2198-9532. Published by Interaktive Demokratie e. V. Available at: [http://www.liquid-democracy-journal.org/issue/4/The\\_Liquid\\_Democracy\\_Journal-Issue004-02-A\\_Finite\\_Discourse\\_Space\\_for\\_an\\_Infinite\\_Number\\_of\\_Participants.html](http://www.liquid-democracy-journal.org/issue/4/The_Liquid_Democracy_Journal-Issue004-02-A_Finite_Discourse_Space_for_an_Infinite_Number_of_Participants.html)

<sup>3</sup><http://dev.liquidfeedback.org/pipermail/main/2015-December/000532.html>

<sup>4</sup>While PostgreSQL supports certain "geometric types" (see <https://www.postgresql.org/docs/9.5/static/datatype-geometric.html>), none of these types are directly suitable for distance calculation on earth or radial searches where latitude and longitude lines are not part of a cartesian coordinate system but where the distance of meridians depend on the latitude.

<sup>5</sup>[http://postgis.net/docs/manual-2.2/postgis\\_installation.html#install\\_requirements](http://postgis.net/docs/manual-2.2/postgis_installation.html#install_requirements)

<sup>6</sup>PostGIS is licensed under the terms of the viral GNU GPL while the LiquidFeedback project depends only on MIT/BSD licensed software.

## 2 Design & Implementation

### 2.1 Removing “membership” in subject areas and implementing a new algorithm for issue admission

As explained in subsection 1.1, LiquidFeedback 3.1 and 3.2 relies on explicit member registration per subject area (“membership in subject areas”) for calculating the first supporter quorum, which is used for issue admission (i.e. deciding which issue may pass from admission to discussion phase). By removing the “membership” information for each member and subject area, an alternative model for issue admission had to be implemented.

#### 2.1.1 Main component of the new admission algorithm

The main component of the new issue admission algorithm (i.e. the algorithm that determines which issues and initiatives pass from admission to discussion phase in LiquidFeedback’s 4-phase model) consists of an SQL view<sup>7</sup> that selects the issue with most supporter voters that is eligible for admission (i.e. not blocked by other recently admitted issues). A draft of this view (and associated helper views) had already been published by the Public Software Group in December 2015.<sup>8</sup> However, this draft was incomplete because it was not integrated with LiquidFeedback’s snapshot system (see following subsection 2.1.2) which is necessary for verifiability of the democratic process by the participants. Beside integrating it with a snapshot system (explained in full detail in the following subsection 2.1.2), other changes needed to be done, such as considering initiative revocations.

#### 2.1.2 Synchronized snapshots

In order to guarantee verifiability of LiquidFeedback’s decision making process, LiquidFeedback utilizes a so-called “snapshots”<sup>9</sup>, which record the initiative’s supporter situation at particular events, such as issue admission or beginning of voting phase. For details, refer to “The Principles of LiquidFeedback”, ISBN 978-3-00-044795-2, section 5.3 (“Availability of data in LiquidFeedback”).

As of LiquidFeedback 3.x, each snapshot only contains information about *one* issue at a given time. The proposal given in “A Finite Discourse Space for an

---

<sup>7</sup>View “issue\_for\_admission”.

<sup>8</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/rev/ca21a3f49e4c](http://www.public-software-group.org/mercurial/liquid_feedback_core/rev/ca21a3f49e4c)

<sup>9</sup>Note that LiquidFeedback’s snapshots and PostgreSQL’s snapshots are two distinct things, even if LiquidFeedback snapshots utilize PostgreSQL’s MVCC system (whose frozen views on data are also referred to as “snapshot”).

Infinite Number Of Participants”<sup>2</sup>, however, explains why a snapshot of several issues at once is necessary:

LiquidFeedback currently relies on a background process, which counts the votes for each issue, one issue after the other. In order to provide a fair process where no issue is favored to any other issue, all issues that may influence each other's dynamic quorum have to be tested for their eligibility at the same time. Since the calculation requires some time to compute, a snapshot must be taken of the supporter situation of all those issues at once.

Such behavior might be achieved by processing all open issues in admission phase at the same time in a single database transaction. In case of the current implementation of LiquidFeedback, however, it might be problematic due to PostgreSQL's way of locking: database locks won't be released until the transaction has finished. Therefore, snapshot synchronization functions should be utilized instead. We refer to section 9.26.5 of the PostgreSQL manual, version 9.4 for further information on this issue.<sup>10</sup>

As a first measure, LiquidFeedback's data structures for holding snapshot information<sup>11</sup> were extended in such way that one snapshot can contain multiple issues.

During implementation of snapshots which are capable to cover multiple issues, cyclic references disclosed an unexpected interdependency with LiquidFeedback's ability to delete old data (e.g. deleting issues that are more than 10 years old): deleting an issue with its associated snapshot data would render snapshots of other issues unusable. Simply using SQL's ON DELETE CASCADE constraints<sup>12</sup> in the cyclic reference between issues and their snapshots would potentially cause all issues to be deleted if an issue refers to a snapshot which in turn refers to several other issues again. In contrast, ON DELETE RESTRICT constraints would render issue deletion impossible. As a solution to this problem, it was necessary to use an ON DELETE SET NULL reference from issues to their respective snapshots that were taken at the end of admission phase.<sup>13</sup>

<sup>10</sup><https://www.postgresql.org/docs/9.4/static/functions-admin.html#FUNCTIONS-SNAPSHOT-SYNCHRONIZATION>

<sup>11</sup>Type "snapshot\_event"; columns "snapshot" and "latest\_snapshot\_event" in table "issue"; tables "direct\_population\_snapshot", "delegating\_population\_snapshot", "direct\_interest\_snapshot", "delegating\_interest\_snapshot", "direct\_supporter\_snapshot" in LiquidFeedback 3.0, 3.1, and 3.2.

<sup>12</sup>Refer to <https://www.postgresql.org/docs/9.5/static/sql-createtable.html> for an overview on referential integrity constraints in PostgreSQL.

<sup>13</sup>For details refer to column "admission\_snapshot\_id" of table "issue" and function "delete\_snapshot\_on\_partial\_delete\_trigger" in the patched core.sql file.

Further obstacles occurred when attempting to utilize PostgreSQL's snapshot synchronization functions<sup>14</sup> to create a consistent snapshot of all issues while trying to avoid locking problems. Section 13.2.2 of PostgreSQL's documentation<sup>15</sup> states for the REPEATABLE READ isolation level that:

UPDATE [or] DELETE, [...] commands behave the same as SELECT in terms of searching for target rows: they will only find target rows that were committed as of the transaction start time. However, such a target row might have already been updated (or deleted or locked) by another concurrent transaction by the time it is found. In this case, the repeatable read transaction will wait for the first updating transaction to commit or roll back (if it is still in progress). [...] if the first updater commits (and actually [...] deleted the row, [...]) then the repeatable read transaction will be rolled back [...] because a repeatable read transaction cannot modify or lock rows changed by other transactions after the repeatable read transaction began.

LiquidFeedback needs to update the supporter counts of suggestions (number of persons in favor or against a suggestion) while taking a snapshot. This is because LiquidFeedback snapshots don't contain information which person was in favor or against each suggestion but only contain information on the initiative level (because initiatives require a certain supporter quorum before being admitted for voting). The supporter counts of suggestions are stored in the suggestion table. Rows in this table, however, may be deleted by users when a suggestion is ranked as "neutral" by all users. Therefore, a serialization conflict may occur between LiquidFeedback's counting process<sup>16</sup> and users who delete suggestions.

This flaw already existed in LiquidFeedback version 3.1 and 3.2 but is negligible if only one issue is taken into consideration at once (the background process can simply retry its attempt to update the issue in the next run). When several issues must be evaluated at once (even if this happens in distinct but synchronized transactions), the probability of a single suggestion being deleted during the process is much higher.

To solve this issue, a new table "temporary\_suggestion\_counts" has been introduced which gets updated instead of the "suggestion" table. The data then gets copied later from the temporary table to the suggestion table.<sup>17</sup>

<sup>14</sup><https://www.postgresql.org/docs/9.5/static/functions-admin.html#FUNCTIONS-SNAPSHOT-SYNCHRONIZATION>

<sup>15</sup><https://www.postgresql.org/docs/9.5/static/transaction-iso.html#XACT-REPEATABLE-READ>

<sup>16</sup>Program "lf\_update" ("lf\_update.c") as shipped by the LiquidFeedback Core distribution.

<sup>17</sup>See function "finish\_snapshot" in the patched core.sql file. Note that it is not necessary to create a temporary table for the issue supporter counts because that information can be fully restored from LiquidFeedback's snapshot tables.

During research on PostgreSQL's capabilities for concurrency control, recent improvements were discovered: PostgreSQL 9.4 provides two new row-level locks: FOR NO KEY UPDATE and FOR KEY SHARE (in addition to the traditional FOR UPDATE and FOR SHARE locks).<sup>18</sup> PostgreSQL 9.3 already prevented non-key-field row updates from blocking foreign key checks<sup>19</sup> (by using a FOR KEY SHARE style lock instead of a FOR SHARE lock on the referenced tables). This improvement allowed to avoid using PostgreSQL's snapshot synchronization functions (unlike previously suggested in "A Finite Discourse Space for an Infinite Number Of Participants"<sup>2</sup>). Writing rows into LiquidFeedback's snapshot tables will not cause a FOR SHARE lock on referenced tables anymore, but only a FOR KEY SHARE lock. Consequently, adjusting the LiquidFeedback Frontend to use a FOR NO KEY UPDATE lock instead of FOR UPDATE locks will allow creation of a snapshot of all issues (or all issues in admission phase) in a single transaction<sup>20</sup> without running into locking problems on heavy-load systems.

Finally, it turned out that always creating one snapshot for all issues isn't feasible because at the end of verification phase (beginning of voting phase), the snapshot data is used to calculate which initiative is eligible for voting. If the calculation of snapshot data is performed for all issues at once, then the users of LiquidFeedback (or any other software component that is connected to LiquidFeedback through its future REST API) would be deterred from performing any action when the verification phase has elapsed but the snapshot data has not been recorded yet. To avoid this problem (i.e. to keep the time delay between end of verification phase and availability of snapshot data as short as possible), two different mechanisms for taking snapshots needed to be provided: the snapshot for all issues in admission phase is taken at once, while snapshots for issues in discussion and verification phase are taken separately for each issue. Furthermore, a grace period was added at the end of admission phase for those issues that have not been admitted yet: only those issues may be canceled where a snapshot has been taken *after* the admission time has ended.

### 2.1.3 Code cleanup (removal of subject area membership related tables)

All structures that were necessary for "membership" in subject areas have been removed.

---

<sup>18</sup><https://www.postgresql.org/docs/9.4/static/explicit-locking.html#LOCKING-ROWS>

<sup>19</sup><https://www.postgresql.org/docs/9.3/static/release-9-3.html>

<sup>20</sup>The snapshot still needs to be finalized for each issue in a different transaction because of updating the "issue", "initiative", etc. tables. See function "finish\_snapshot" in the patched core.sql file.

### 2.1.4 Publication of results

All necessary adjustments to the data structures as well all functions in LiquidFeedback's core.sql file for background counting could be implemented as of May 27, 2016. The work has been submitted to the Public Software Group e. V. and was published on May 29, 2016 in the source code repository of LiquidFeedback Core<sup>21</sup> under the terms of the MIT-License<sup>22</sup>. The background update program ("lf\_update") has not been updated yet because it is not necessary for testing and/or development purposes at this stage.<sup>23</sup>

## 2.2 Geospatial data support

The work tasks explained in subsections 1.2 and 1.3 are highly interrelated and have thus been implemented concurrently. Their implementation will be explained in the following.

### 2.2.1 Considering PostGIS for geospatial support

PostGIS (see <http://www.postgis.net/>) is a 3<sup>rd</sup> party extension for the PostgreSQL database server. Using PostGIS for the geospatial functions of LiquidFeedback was finally ruled out due to the following considerations:

1. Using PostGIS would introduce a long chain of software dependencies compared to the current lightweight approach of LiquidFeedback. The dependencies of PostGIS include:
  - The PROJ.4 Cartographic Projections Library<sup>24</sup>
  - The GEOS Geometry Engine<sup>25</sup>
  - The XML C parser and toolkit of Gnome (libxml2)<sup>26</sup>
  - A JSON library "json-c"<sup>27</sup>

---

<sup>21</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/rev/3e28fd842354](http://www.public-software-group.org/mercurial/liquid_feedback_core/rev/3e28fd842354)

<sup>22</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/file/3e28fd842354/LICENSE](http://www.public-software-group.org/mercurial/liquid_feedback_core/file/3e28fd842354/LICENSE)

<sup>23</sup>The function "check\_everything" can be used to trigger the background counting process in a single transaction. While this is sufficient for testing purposes, the changes to this function need to be adopted in the "lf\_update" program as well, since this program utilizes different transactions with different transaction isolation levels for each step of the counting process, which is necessary to avoid locking problems on production systems (see also footnote 20).

<sup>24</sup><https://github.com/OSGeo/proj.4/>

<sup>25</sup><https://trac.osgeo.org/geos/>

<sup>26</sup><http://xmlsoft.org/>

<sup>27</sup><https://github.com/json-c/json-c/>

- The GDAL Geospatial Data Abstraction Library<sup>28</sup>

For an exhaustive list of all dependencies of PostGIS, refer to:

[http://postgis.net/docs/manual-2.2/postgis\\_installation.html#install\\_requirements](http://postgis.net/docs/manual-2.2/postgis_installation.html#install_requirements)

The LiquidFeedback Core, in contrast, has no further dependencies (except PostgreSQL and a C-compiler).

2. Many of PostGIS' features are unnecessary for LiquidFeedback. At the same time, the number of functions that actually support the WGS-84 ellipsoid (PostGIS's "Geography type") is limited. The PostGIS 2.2 manual states in section 4.2:<sup>29</sup>

Because the underlying mathematics is much more complicated, there are fewer functions defined for the geography type than for the geometry type.

3. PostGIS is licensed under the terms of the GNU General Public License, version 2, which is a viral license. The PostGIS license FAQ<sup>30</sup> states the following:

Question: I am releasing software that uses PostGIS, does that mean my software has to be licensed using the GPL like PostGIS? Will I have to publish all my code if I use PostGIS?

Answer: Almost certainly not. [...] your software can use a PostgreSQL/PostGIS database as much as it wants and be under any license you like.

While the license FAQ covers the *usage* of PostGIS, it does not cover other forms of commercial exploitation such as renting integrated software solutions or sale of hardware appliances. It also does not cover any issues potentially introduced when incorporating the software into a web service that has to be sublicensed under the terms of another viral license, e.g. the GNU Affero GPL, version 3.<sup>31</sup>

Notwithstanding, the legal interpretation found on the homepage of the PostGIS project will not be binding for all potential legal disputes. Taking

---

<sup>28</sup><https://trac.osgeo.org/gdal/>

<sup>29</sup>[http://postgis.net/docs/manual-2.2/using\\_postgis\\_dbmanagement.html#PostGIS\\_Geography](http://postgis.net/docs/manual-2.2/using_postgis_dbmanagement.html#PostGIS_Geography)

<sup>30</sup>[http://postgis.net/docs/manual-dev/PostGIS\\_FAQ.html#license\\_faq](http://postgis.net/docs/manual-dev/PostGIS_FAQ.html#license_faq)

<sup>31</sup>To name an example for possible licence incompatibilities, we refer to Improve My City, which is has been licensed under the terms of the GNU Affero GPL, version 3, even though in this case a re-licensing was considered by the author (refer to D1.1 in that matter).

a look at the original wording of the GNU General Public License, version 2, we find the following clause:

If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Because of these reasons, PostGIS was not considered suitable when extending LiquidFeedback's backend for geospatial support.

It should be noted that our research discovered hidden GPL dependencies (i.e. not mentioned in D1.1) in other software components of the other consortium partners. Subsection 3.2 will get back to this issue.

### 2.2.2 Vincenty's formulae

After ruling out PostGIS, alternatives for spatial indices and radial searches had to be considered. One method to calculate distances on the WGS-84 reference ellipsoid are Vincenty's formulae<sup>32</sup> (second method for computing geographical distances and azimuth between two given points). Spatial indexing could then be implemented by using bounding boxes that are calculated based on a spherical earth<sup>33</sup> with a margin that compensates the error due to using a sphere instead of an ellipsoid.

However, Vincenty's formulae are iterative methods that are complex to implement and expensive in terms of computing time.

### 2.2.3 A lightweight alternative: 2-dimensional truncated Taylor series

A lightweight alternative to Vincenty's formulae is to use a geographic projection to a 2-dimensional plane and to calculate distances simply using the Pythagorean theorem. This method is a common technique that is also used by PostGIS (in form of the "Geometry type") and even supported in a basic manner by PostgreSQL itself. The PostGIS manual recommends to use geographic projections

<sup>32</sup>[https://en.wikipedia.org/wiki/Vincenty%27s\\_formulae](https://en.wikipedia.org/wiki/Vincenty%27s_formulae)

<sup>33</sup>A formula for the sphere is found in: Bronstein, Semendjajew, Musiol, Mühlig: Taschenbuch der Mathematik. 6<sup>th</sup> edition, page 182 ("Tangierpunkte"). ISBN 978-3-8171-2006-2.

if the expected working area of the built application is small:<sup>34</sup>

The type you choose should be conditioned on the expected working area of the application you are building. Will your data span the globe or a large continental area, or is it local to a state, county or municipality?

If your data is contained in a small area, you might find that choosing an appropriate projection and using GEOMETRY is the best solution, in terms of performance and functionality available.

Assuming a projection with the parallels (circles of latitude) being orthogonal to the meridians (lines of longitude), and optimizing the projection for distances close to one point  $(\phi_0, \lambda_0)$  (where  $\phi_0$  denotes the latitude and  $\lambda_0$  denotes the longitude), then this results in the following formula for a distance  $s$  between two points  $(\phi, \lambda)$  and  $(\phi_0, \lambda_0)$  on earth:

$$s^2 \approx \tilde{s}^2 = t_{20}(\phi - \phi_0)^2 + t_{02}(\lambda - \lambda_0)^2$$

with

$$t_{20} = \frac{\frac{\partial^2}{\partial \phi^2}(s^2)}{2} \Big|_{\phi=\phi_0, \lambda=\lambda_0}$$

$$t_{02} = \frac{\frac{\partial^2}{\partial \lambda^2}(s^2)}{2} \Big|_{\phi=\phi_0, \lambda=\lambda_0}$$

where  $s$  is the distance on the surface,  $\tilde{s}$  is the approximated distance on the surface for points close to  $\phi_0$  and  $\lambda_0$ , and  $t_{ij}$  are Taylor coefficients.

We can generalize this idea by using the following alternative definition for  $\tilde{s}$  that contains more terms of the Taylor series:

$$s^2 \approx \tilde{s}^2 = \sum_{i=0}^{i_{\max}} \sum_{j=0}^{j_{\max}} \underbrace{\frac{(\frac{\partial}{\partial \phi})^i (\frac{\partial}{\partial \lambda})^j (s^2) \Big|_{\phi=\phi_0, \lambda=\lambda_0}}{i! j!}}_{=: t_{ij}} (\phi - \phi_0)^i (\lambda - \lambda_0)^j$$

Formulae for the coefficients  $t_{ij}$  for  $i_{\max} = j_{\max} = 2$  have already been determined in the year 2012. A corresponding paper that includes the mathematical computations is available at: <http://www.public-software-group.org/pub/papers/taylor-distance-2012-12-31.pdf>

The above mentioned paper also contains methods of calculating latitude/longitude bounding boxes for spatial index lookup. Both the distance calculation as well as the bounding box calculation have been successfully implemented in PL/pgSQL (see section 2.2.13 for a summary of the published results on geospatial issues).

<sup>34</sup>[http://postgis.net/docs/manual-2.2/using\\_postgis\\_dbmanagement.html#PostGIS\\_GeographyVSGeometry](http://postgis.net/docs/manual-2.2/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry)

## 2.2.4 Using PostgreSQL's geometric data types, functions, and operators

With the help of a Taylor approximation for distance calculations as outlined in the previous subsection 2.2.3, it has been possible to use PostgreSQL's built-in geometric data types by utilizing the following mapping between geographic objects and geometric data types:

Geographic object	Data type in PostgreSQL
Point	POINT or PATH with one point only
Path	PATH (open)
Closed path (not filled)	PATH (open), with last point equal to first
Filled polygon	PATH (closed) or POLYGON
Region consisting of multiple polygons which may contain holes	POLYGON [ ]
Circle (e.g. for radial searches)	POLYGON (where necessary, see below)

PostgreSQL's CIRCLE data type is not usable because it assumes a Cartesian coordinate system.<sup>35</sup> Instead, a polygon can be used that is created with the help of the Taylor approximation explained in the previous subsection 2.2.3. The more edges/vertices the polygon has, the smaller is the error.<sup>36</sup> However, in case of radial searches of points, a geometric representation of the search circle is not necessary: a bounding box for latitude and longitude can be used instead and the positive matches can be filtered with the Taylor approximation mentioned in subsection 2.2.3.<sup>37</sup> Regions can be modeled as an array of polygons (POLYGON [ ]). A point is contained in the region if the number of polygons in which it is contained is odd.<sup>38</sup>

As we will later see in subsection 2.2.12, only a limited range of geographic object types was needed for LiquidFeedback at this stage.

## 2.2.5 Utilizing PostgreSQL's GiST and SP-GiST index methods

As of PostgreSQL version 9.5, there are several different indexing methods available for the geometric data types:

<sup>35</sup>The same holds for several geometric operators of PostgreSQL, like the distance operator <-> or the "closest point" operator #. Note that several of those operators and functions do not work as expected/documented anyway, as a review of the "geo\_ops.c" file of PostgreSQL's source code revealed.

<sup>36</sup>See function "polysearch" in the patched core.sql file.

<sup>37</sup>See function "geosearch" in the patched core.sql file.

<sup>38</sup>See function "scattered\_polygon\_contains\_point" in the patched core.sql file.

- GiST index<sup>39</sup> for BOX, CIRCLE, POINT, and POLYGON,
- SP-GiST index with “quad\_point\_ops” (quadtree) operator class for POINT,
- SP-GiST index with “kd\_point\_ops” (kd-tree) operator class for POINT.

For indexing points, the quadtree implementation using the SP-GiST framework has been chosen.<sup>40</sup> For all other geographic objects, GiST indices must be used. Values of type “PATH” can be converted to a “POLYGON” prior to indexing.<sup>41</sup>

Because quadtrees are sensible to different stretching in each dimension (i.e. searches should be nearly radial in the underlying coordinate system and not be stretched in either X- or Y-direction), the latitudes and longitudes of each indexed point in degree (from  $-90^\circ$  to  $+90^\circ$  or, respectively,  $-180^\circ$  to  $+180^\circ$ ) get converted to a tuple  $([-1, +1], [-1, +1])$  prior to indexing.<sup>42</sup> While this stretches the meridians by a factor of 2 at the equator, it compensates the compression of the meridians near the poles (which would otherwise be approx.  $\cos(75^\circ) \approx \frac{1}{4}$  at latitude  $75^\circ$  and only  $\frac{1}{2}$  with the compensation).

Examples on index usage are given in the patched core.sql file<sup>43</sup> in the section “spatial indexing and distance calculation” (lines 49 through 625) that also defines the corresponding functions.

### 2.2.6 Numerical stability

The background paper from 2012 on approximating the distance on earth using a 2-dimensional Taylor series (see subsection 2.2.3) only covered aspects of analysis but not of numerics. Special considerations were taken to avoid floating point errors such as taking square roots from numbers that are almost zero but slightly negative, etc.

### 2.2.7 Poles and the 180<sup>th</sup> meridian

In order to allow for a general usage of the created system, issues with the 180<sup>th</sup> meridian have been considered and taken care of during implementation. Radial searches close to the poles (latitudes greater than  $84^\circ$  or smaller than  $-84^\circ$ ),

---

<sup>39</sup>With R-tree behavior, see “src/backend/access/gist/gistproc.c” in PostgreSQL 9.5.

<sup>40</sup>As already mentioned in footnote 35, the existing distance operator can't be used, which reduces the advantages of GiST indices. Nonetheless, using a GiST index is still possible.

<sup>41</sup>See “polyindex” functions in the patched core.sql file as well as section 11.7 on “indexes on expressions” in the PostgreSQL 9.5 manual: <https://www.postgresql.org/docs/9.5/static/indexes-expressional.html>

<sup>42</sup>See function “geoindex” in the patched core.sql file.

<sup>43</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/file/3e28fd842354/core.sql](http://www.public-software-group.org/mercurial/liquid_feedback_core/file/3e28fd842354/core.sql)

however, are not yielding correct results as of the current implementation. Because the (truncated) Taylor series works for local searches only (e.g. in the scope of a single country or the European Union but not across continents), excluding the poles does not impose further restrictions than the Taylor approximation does anyway.

Future work could lift these restrictions.

## 2.2.8 Examples of the accuracy of the truncated Taylor series

We consider the following locations:

Location	Sym.	Latitude	Longitude
Berlin (Brandenburg Gate)	B	52.5162746 N	13.3777040 E
London (Big Ben)	L	51.5007292 N	0.1246254 W
Paris (Eiffel Tower)	P	48.8583701 N	2.2944813 W
Rome (Colosseum)	R	41.8902102 N	12.4922309 W
Moscow (Red Square, Mausoleum)	M	55.7537117 N	37.6198846 E
Longyearbyen (harbour)	X	78.2289157 N	15.5994530 W

The implemented 2-dimensional Taylor series for distance calculation then results in the following relative errors:

Error in ‰	To B	To L	To P	To R	To M	To X
From B (Berlin)	0	+1.40	+1.41	+1.96	+4.42	−6.95
From L (London)	+1.39	0	+0.15	+0.08	+11.09	−3.97
From P (Paris)	+1.29	−0.21	0	+0.09	+10.72	−3.94
From R (Rome)	+0.63	−1.23	−0.66	0	+10.58	−3.03
From M (Moscow)	+4.48	+11.20	+11.15	+13.26	0	−2.01
From X (Longy.)	+1.82	+1.34	+1.55	+2.25	+11.65	0

As expected from a Taylor approximation, the error is smaller if the locations are closer to one another (i.e. if the argument to the Taylor series  $\phi, \lambda$  is close to the origin point  $\phi_0, \lambda_0$  for which the coefficients  $t_{i,j}$  have been determined).

For the currently planned use cases of WeGovNow (application inside a single city or borrow), the distances are indeed much shorter, and thus the relative (and absolute) error will be smaller than the given examples. But even considering a Europe-wide application, the average error is still smaller than the expected error when modeling the earth as sphere (in which case errors of up to  $\pm 5\%$  would have to be expected, even in case of very short distances).

It should be noted that the error is not symmetric (e.g. the error when calculating the distance from Berlin to London is different from the error when

calculating the distance from London to Berlin). This asymmetry is due to the nature of the Taylor approximation. It is computationally simple to calculate distances from a single source point ("From") to hundreds or thousands of destination points ("To"),<sup>44</sup> but performing the calculation in the opposite direction has a higher computational complexity (because the Taylor coefficients would need to be re-calculated for each pair of locations). The following subsection 2.2.9 will comment on the implications of the lack of reversal-symmetry.

### 2.2.9 Reversal-symmetry anomalies and large-area applications

As shown in the previous subsection 2.2.8, the Taylor approximation method lacks reversal-symmetry (i.e. the distance calculated from one location 'A' to another location 'B' is not exactly the same as the distance from location 'B' to location 'A'). The relative difference is generally very small and tends towards zero if the distance of the locations is very small (e.g. in the same city). Nevertheless, there is a (small) possibility of end-users being confused by this behavior. Consider an initiative 'I' with a listing of "all other initiatives within 10km radius". Let further assume that an initiative 'J' is contained in that list. Due to the lack of reversal-symmetry, it could happen that while 'J' is in the list of 'I', the reverse is not true.

While the probability tends towards zero if the points of interest are close to one another (e.g. in the same city), the problem cannot be solved completely without sacrificing the advantages of the Taylor approximation (which is an efficient way to mass-calculate distances from a single point to many other points in the database). Nevertheless, further research may reveal alternatives to the currently Taylor series based approach.

An alternative approach might also solve the problem of large-area applications in the future, where, for example, the distance over the North pole may be significantly closer than walking/flying along the circles of latitude.

#### 2.2.10 Performance

While there was no formal benchmark, some tests of the SP-GiST quadtree index for points and the Taylor series based radial search revealed an amazing performance considering that all functions have been implemented in SQL and PL/pgSQL.

The speed might be improved further by providing C-language based functions. However, there is no doubt that the current implementation is sufficiently fast.

---

<sup>44</sup>See function "geodist" in the patched core.sql file.

### 2.2.11 Incremental neighbor search

The created implementation only offers radial searches with a fixed radius per query. The LiquidFeedback API, however, should also offer an incremental search where all initiatives (or suggestions) close to a given location are listed ordered by distance (e.g. “give me those 100 initiatives which are closest to the Brandenburg Gate”).

The problem can be easily solved by exponentially increasing the search radius, e.g. 1 km, 2 km, 4 km, 8 km until enough initiatives (or suggestions) are found. This can be implemented by the LiquidFeedback API layer and is an implementation detail that is not exposed to the clients of LiquidFeedback’s API. Because of the exponential nature, the overhead is of logarithmic order.<sup>45</sup>

### 2.2.12 Adding data fields for geo-tagging and applying the geospatial functions to LiquidFeedback’s backend

The data structures to be stored for each user-generated datum and other entities (e.g. subject areas) in LiquidFeedback had to be determined. Beside the member profile, there are 3 possible user-generated entities that might require geographical information:

- issues (stored in table “issue”),
- initiatives (stored in tables “initiative” and “draft”),
- suggestions (stored in table “suggestion”).

For the same reasons why issues have no name, they also should not have any other user-generated information on them (beside that information which is part of the initiatives, drafts, or suggestions being part of the issue). Giving users the option to modify issue-specific data would contradict the collective moderation process of LiquidFeedback. Refer to “The Principles of LiquidFeedback” (ISBN 978-3-00-044795-2), sections 4.3 (“Collective moderation”) and 4.4 (“Unlabeled groups of alternative initiatives”) for that matter.

In regard to initiatives, it is thinkable that geographic data of an initiative gets updated within the discussion process. Therefore, it is appropriate to assign any geographic information to the initiative’s draft (table “draft”) instead of the initiative table.<sup>46</sup>

---

<sup>45</sup>A better alternative would be to provide an own implementation of a geographic data type in PostgreSQL which supports GiST indexing with nearest neighbor search (on the WGS-84 ellipsoid). Future research may be of interest.

<sup>46</sup>A copy of the most recent data, however, is stored in the initiative table to simplify queries on current data only.

Several different approaches of allowed geographic types for initiatives and suggestions have been considered:

- storing one point per user-generated entity,
- storing multiple points per user-generated entity,
- storing multiple points or other geographical structures (pathes, regions, etc.) per user-generated entity.

However, keeping the requirement of collective moderation in mind, it seemed infeasible to allow users to tag their content with a potentially unlimited number of geographic objects (or areas of arbitrary size), unless further research on a fair ranking algorithm for geographical searches and/or map display has been done.

As a result, it was decided to start with a conservative approach where only a limited number  $N_{\max}$  of points (i.e. coordinates given by latitude and longitude) may be stored for each user-generated datum. This restriction may be lifted after future research.

As for user profiles, a value of  $N_{\max} = 1$  seems legitimate. For initiatives (drafts) and suggestions, we chose a value of  $N_{\max} = 2$ . This ensures that users may properly tag proposals like “move the annual event X from point A to point B” or “improve the street C to reduce traffic at street D”, etc. At the same time, it is not possible to cover large areas with the given tagging mechanism: users will have to decide on a center tag instead of marking huge areas with multiple points (possible if  $N_{\max} \geq 3$ ) and/or polygons.

Beside these considerations for user-generated content, the following two administratively created entities received a geographical datum as well:

- organizational unit,
- subject area.

The POLYGON [ ] data type (array of POLYGONS) has been selected for these entities in order to allow for marking regions consisting of multiple polygons which may contain holes (see subsection 2.2.4). The LiquidFeedback frontend and API may use these regions as a “whitelist” when deciding if a geographic marker may be set for a particular initiative or suggestion (i.e. it could be forbidden to set a marker to South America if the organizational unit or subject area refers to a London borough).

### 2.2.13 Publication of results

The Taylor series based approach has been implemented successfully and published as part of the patch for LiquidFeedback’s core.sql file. The same holds for

the the necessary data structures and spatial indices for geo-tagging of member profiles, initiatives/drafts, and suggestions (as well as storing regions for organizational units and subject areas). The work has been submitted to the Public Software Group e. V. on May 27, 2016 and was published on May 29, 2016 in the source code repository of LiquidFeedback Core<sup>21</sup> under the terms of the MIT-License<sup>22</sup>.

## 3 Conclusion & Outlook

### 3.1 Fulfillment of planned work tasks

All planned tasks could be accomplished, with two exceptions:

- The background counting program (“lf\_update”) has not been adjusted yet (see subsection 2.1.4).
- The implemented geospatial functions have potential for improvement. There is a tiny chance<sup>47</sup> of reversal-symmetry anomalies with the implemented Taylor series based approach for distance calculation (see subsection 2.2.9) and the algorithm is not suitable for global application yet (i.e. applications that span several continents, see also subsection 2.2.9). Also the nearest neighbor search could be improved.<sup>45</sup>

As for the first issue, it should be noted that LiquidFeedback’s “check\_everything” function has been updated which performs the very same tasks as the background counting program “lf\_update”, except that the locking and transaction isolation model is not suitable for productive use cases.<sup>48</sup> For matters of testing and development, however, the implemented solution is already feasible.

The implemented approach for geo-tagging and geospatial indexing is feasible for the planned areas of application (London Southwark, Torino, San Donà di Piave). However, keeping future applications in mind, further work should be invested on solving the reversal-symmetry issue in case of applications spanning the whole European Union or multiple continents.

---

<sup>47</sup>The probability tends towards zero for small-area applications such as usage within a borough of a city.

<sup>48</sup>This is because PostgreSQL enforces a function to be executed within a single transaction, i.e. it is not possible to use BEGIN and COMMIT/ROLLBACK statements within an SQL function that is stored in the database (stored procedure).

### 3.2 Restrictive/viral licenses discovered in dependencies

Analysis of open source software licenses discovered that report D1.1 missed information on GPL licensed software dependencies. Subsection 1.2.2 (“GeoKey and Community Maps”) of D1.1 does not mention any GPL licenses when answering question #13 on page 23:

Answer #13: GeoKey – Apache 2.0. All frameworks/libraries/plugins used have either MIT, or BSD, or Apache 2.0 licenses, Pillow uses PIL license.

The answer neither mentions any proprietary licenses nor any GPL licensed dependencies. However, this appears to be not true because our research revealed PostGIS (which is GPL licensed) to be a mandatory dependency of UCL’s GeoKey.<sup>49</sup>

It should be noted that the same applies to FirstLife.<sup>50</sup> However, UNITO noted that FirstLife only depends on a few features of PostGIS and that it would be generally desirable to get rid of PostGIS as a mandatory dependency.

All consortium partners should reaudit their software dependencies in order to track down any previously missed GPL licensed and/or proprietary software components.

### 3.3 Remaining tasks

1. The background counting program (“lf\_update”) of LiquidFeedback needs to reflect the implemented changes for the new issue admission process (see subsection 2.1).
2. Keeping future applications in mind, it would be beneficial to improve the implemented methods for geospatial indexing and radial searches by providing an alternative formula for distance calculation on the WGS-84 ellipsoid that could serve as a compromise between the complexity of Vincenty’s formulae and the simplicity of local Taylor approximation. Utilizing the nearest neighbor search capabilities of PostgreSQL’s GiST indexing framework would be another improvement. (optional)

---

<sup>49</sup>Refer to the installation instructions at <http://geokey.org.uk/help/how-to-install.html> or to the source code package of GeoKey.

<sup>50</sup>Refer to D1.1, questions 3 and 13 on pages 19 and 20 respectively.

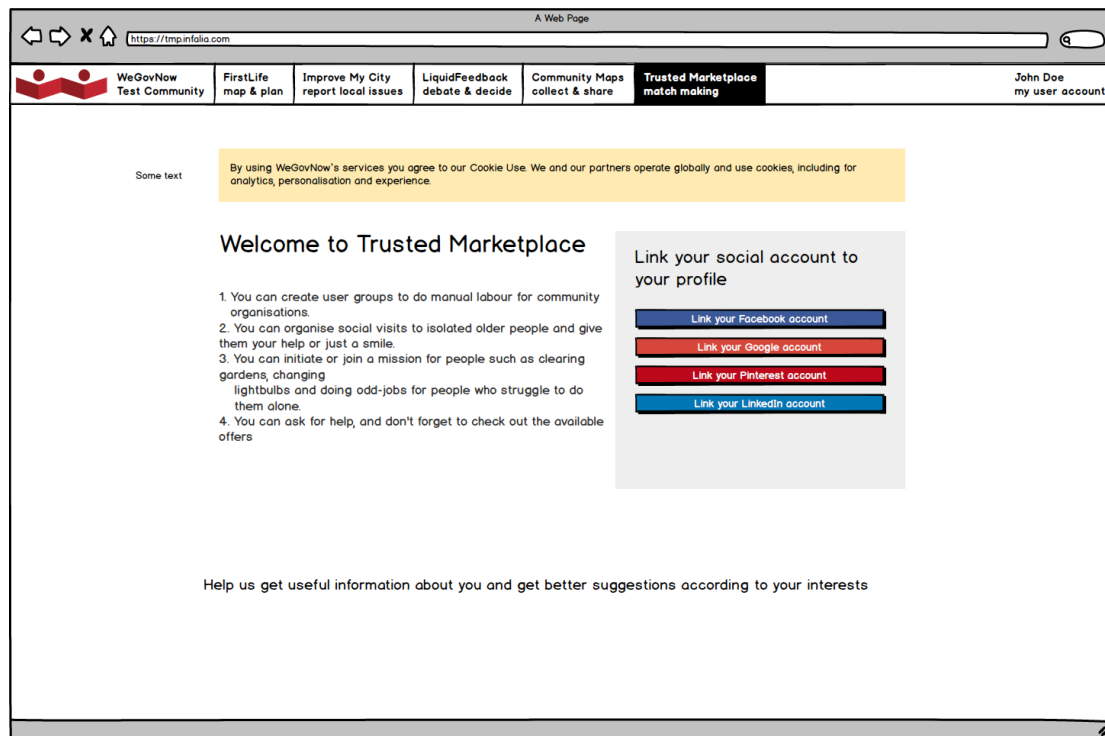
3. As explained in subsection 3.2, a complete dependency and license review of all WeGovNow software components should be done, since errors have been found in D1.1. Refer to the previous subsection 3.2 for details.

© 2016 FlexiGuided GmbH, Berlin

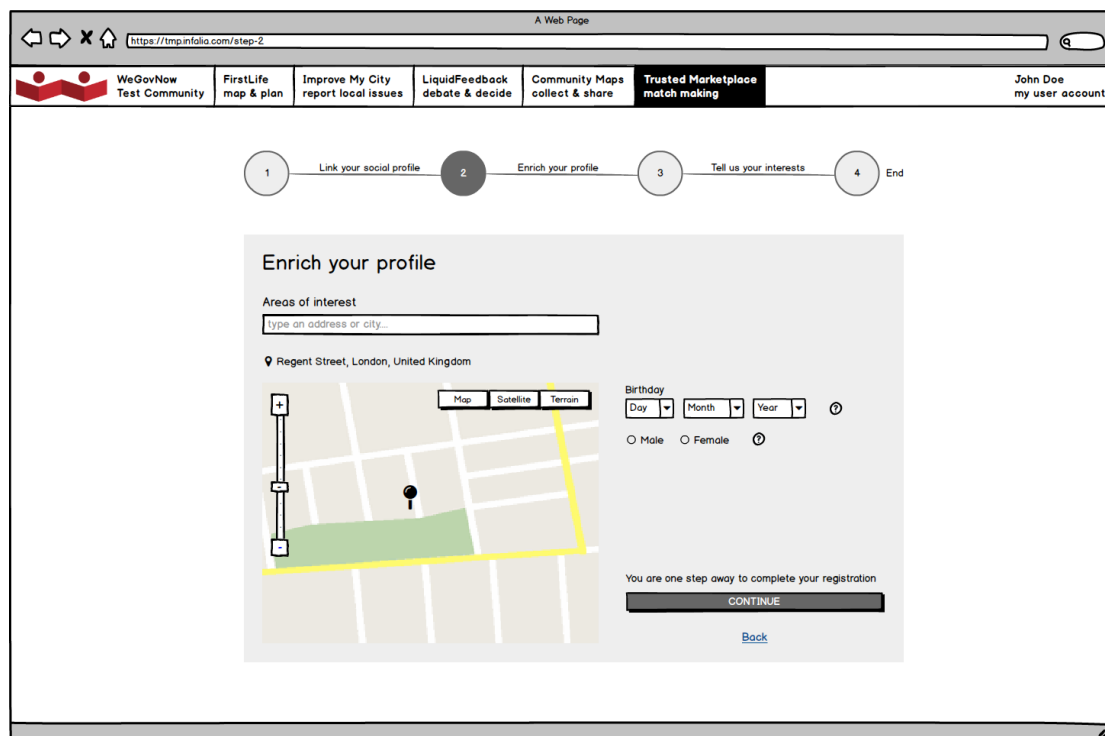
## **Appendix 3.7**

### **Trusted Marketplace Mockups**

## Trusted Marketplace mock-ups



### Link social media accounts



### Profile enrichment

A Web Page

https://tmp.infalia.com/step-3

WeGovNow Test Community | FirstLife map & plan | Improve My City report local issues | LiquidFeedback debate & decide | **Trusted Marketplace match making** | John Doe my user account

1 Account registration — 2 A little more details — 3 Tell us your interests — 4 End

### Tell us some of your interests

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Home decoration	Design	Art	Fashion	Photography
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Technology	Travel	Architecture	Education	Health

**Finish**

WeGovNow

Test Community

FirstLife

map & plan

Improve My City

report local issues

LiquidFeedback

debate & decide

Community Maps

collect & share

Trusted Marketplace

match making

John Doe

my user account

Joe Doe

Trusted score 70%

Post an Offer/Demand

Dashboard

View Offers/Demands

Personalised timeline

Notifications

Sign out

Settings

Help

Settings

Basic info

Work & Education

Interests

Social networks

Email

joedoe@mail.com

Phone number

6944444444

Birthday

21

May

1982

Name day

21

May

Gender

Male

Female

Languages

English

Italian

About you...

Tell us about you...

Cancel

Save settings

WeGovNow  
Test Community

FirstLife  
map & plan

Improve My City  
report local issues

LiquidFeedback  
debate & decide

Community Maps  
collect & share

Trusted Marketplace  
match making

John Doe  
my user account

Joe Doe  
Trusted score 70%

Post an Offer/Demand

Dashboard

View Offers/Demands

Personalised timeline

Notifications 1

Sign out

Settings Help

## Settings

Basic info
Work & Education
Interests
Social networks

### WORK

**Infalia Private Company** Current

City/Town: Thessaloniki, Greece

Role: Web developer

**Some Company**

City/Town: London, UK

Role: Web developer

### EDUCATION

**Aristotle University of Thessaloniki**

City/Town: Thessaloniki, Greece

Studies: Information Technology

### SKILLS

Web development

Server administration

UX Design

## Settings Basic info (more details)

WeGovNow  
Test Community

FirstLife  
map & plan

Improve My City  
report local issues

LiquidFeedback  
debate & decide

Community Maps  
collect & share

Trusted Marketplace  
match making

John Doe  
my user account

Joe Doe  
Trusted score 70%

Post an Offer/Demand

Dashboard

View Offers/Demands

Personalised timeline

Notifications 1

Sign out

Settings Help

## Settings

Basic info
Work & Education
Interests
Social networks

### GENERAL INTERESTS

Photography

Education

Travel

Health

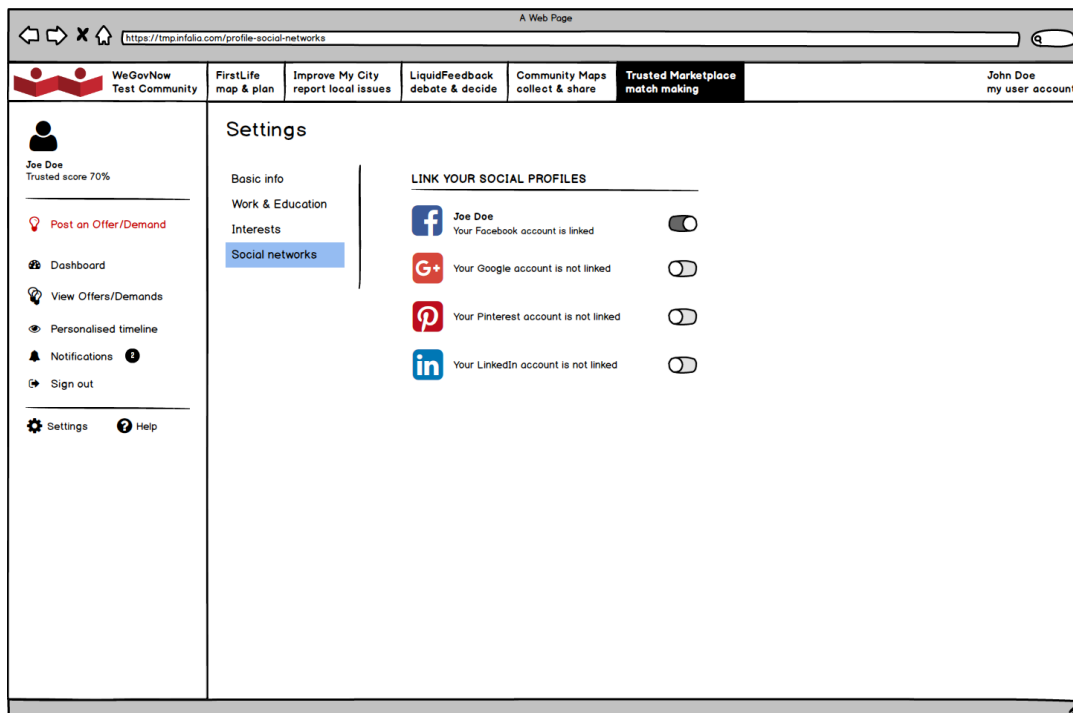
### AREAS OF INTEREST

Regent Street, London, United Kingdom

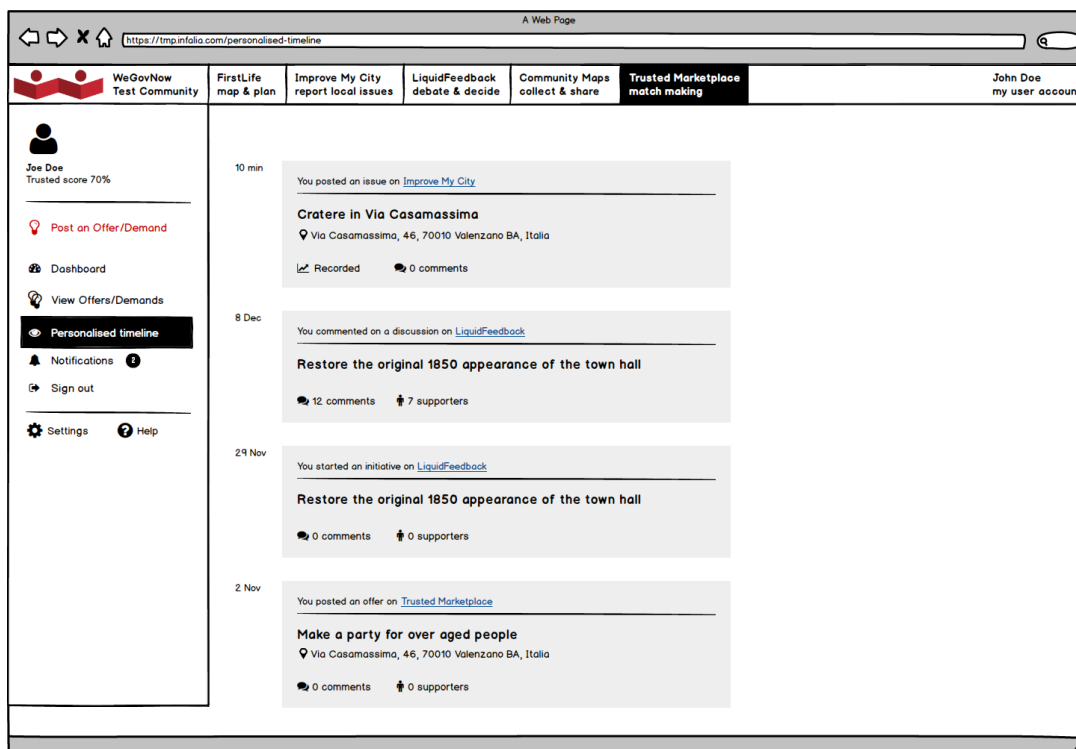
Hyde Park, London, United Kingdom

Dare Valley Country Park, Wales, United Kingdom

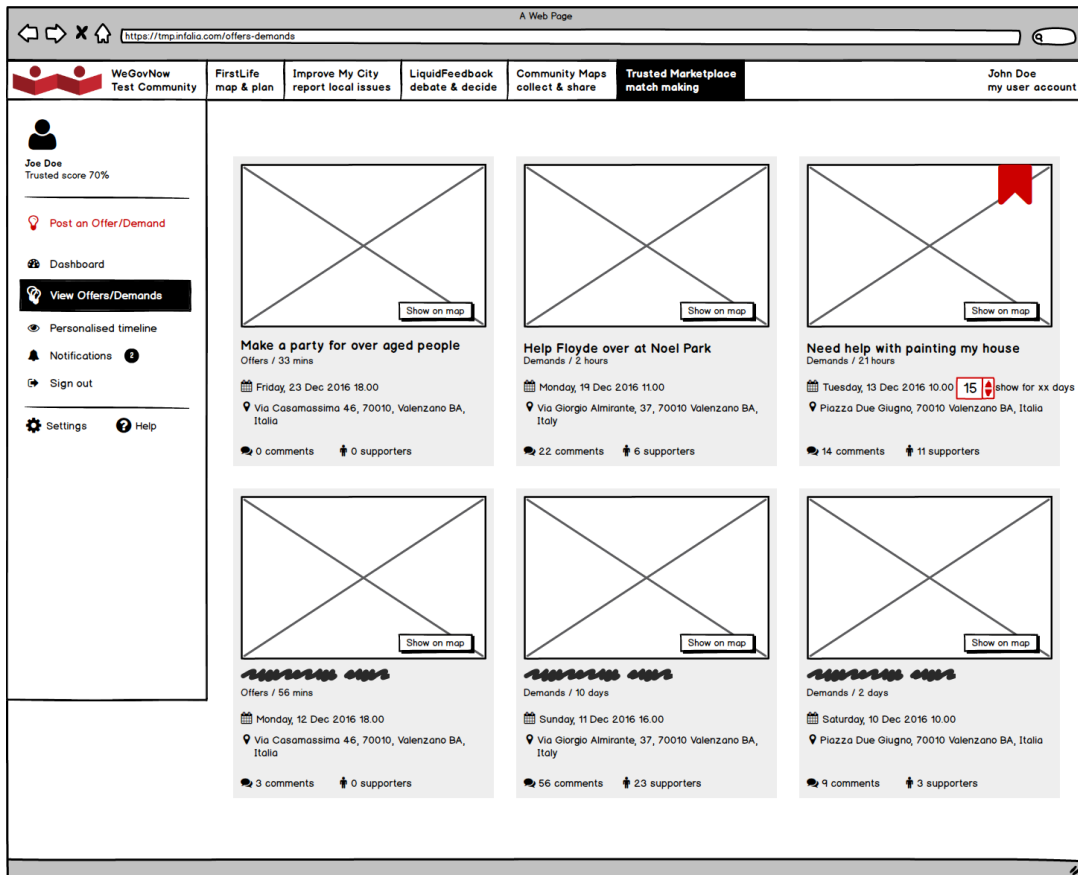
User interests are always editable



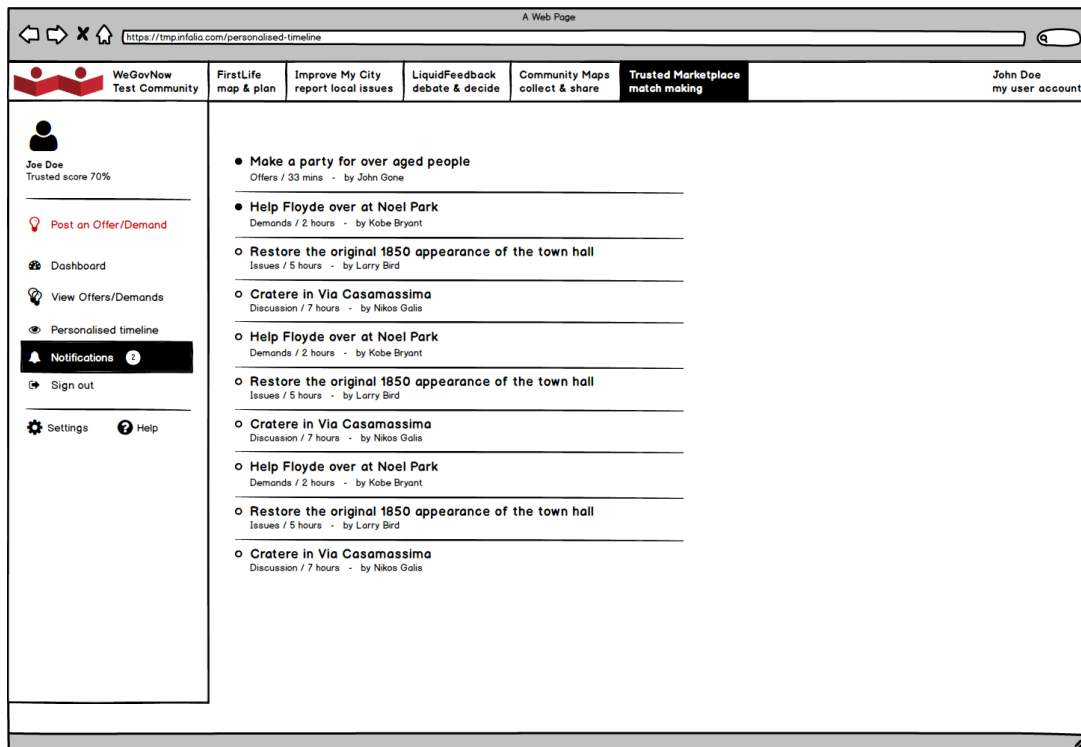
User can link/unlink social accounts easily



Presentation of the personalised timeline



## Offers and Demands module



## Overview of the received notifications



## **Appendix 5.1**

### User Requirements

## Appendix 5.2 - User Requirements

The requirements were extracted from the scenarios and the interviews using thematic analysis of the word documents and the transcriptions from the skype interviews with the municipalities, as well as from information derived from the one-day on site Service Scenarios and Requirements Gathering Exercises carried out with San Dona, Torino and Southwark councils. The template that was created during these exercises was then used to collect and extract the requirements from the scenarios in a more detailed manner for the technical teams.

A spreadsheet was created for each municipality consisting of the requirements of each service scenario. Fields include a unique requirement ID, which was going to be created when all the requirements were merged together, a user requirement ID for the specific council, consisting of the service scenario name and the requirement number, a critical scale field, indication whether the requirement in hand is functional or non-functional, requirement description, times mentioned, a field for user roles concerned and the proposed owner for the specific requirement.

A total of 309 user requirements were extracted from the scenarios provided by the municipalities. Each user requirement was assigned to the use case it was extracted from and the proposed owner (in WeGovNow's case, the municipality).

### Requirements Count

Southwark	167
Turin	75
San Dona Di Piave	67

The following table is the template used for listing the requirements gathered for each use case scenario. The subsequent paragraphs describe in detail each of the table fields (columns).

Requirement ID	Critical Scale	Function Type	Requirement	Frequency	User Roles Concerned	Owner

### Requirement IDn

Uniquely identifies the requirement in the context of each municipality. The requirement ID indicates the number of the use case from which the requirement was taken as well as the initials of the title of the scenario. It is case study specific with plans to be changed into a unique id for a global list of requirements.

### Critical Scale

Refers to the importance of the requirements in regards to a critical scale built for requirements severance. MoSCoW and a scale of severity (1-5) were proposed to be applied with the help of the technical team.

### Function Type

Either F (Functional) or NF (Non-Functional): Functional requirements define a function of the system, while non-functional requirements refer to constraints on the implementation or design of the system.

### Requirement

The requirement field contains the requirement in the form of a short and concise sentence. It consists information about the action that should be made. The format of this field is shown as the example below:

The system must allow authorised staff to update documents, input text and upload pictures.
---

### Frequency

Frequency refers to how many times each requirement was mentioned both within the proposed owners (the specific case-study in question) and between use cases and municipalities. It was added on the premise of helping identify each requirement's place in the critical scale.

### User Roles Concerned

A reference to which user roles will affect each action (requirement).

### Owner

The proposing municipality.

The lists of user requirements for all three municipalities can be found following this link:

[https://docs.google.com/spreadsheets/d/1AOxT9Slm3W1vAg\\_nqjgD\\_p33MPzeo85cRxgS93m8cTE/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1AOxT9Slm3W1vAg_nqjgD_p33MPzeo85cRxgS93m8cTE/edit?usp=sharing)

## **Appendix 5.2**

### Assessment Grid

## Appendix 5.2 - Assessment Grid

The development of the platform should follow the analysis of the outcome of the requirement elicitation and of the scenarios developed by the municipality involved in WeGovNow. The service scenarios provide contextual information about the service the municipalities wish to provide, and use cases grounded on their understanding of the features of the software components, functional and nonfunctional required are extracted and normalised on the bases of the use cases as mentioned in the service scenarios.

The extraction of the development cannot be done from the output of the engagement activities but requires two preliminary steps:

1. **Analysis and clustering of the service scenarios**, each scenario is analysed enquiring the proposers in order to design the general process behind the scenarios. The general process is used as framework to represent the scenario in terms of actors (roles), phases and actions (features).
2. **Assessment and mapping of the actions**, first the actions are evaluated in terms of existing features, scope of the projects and scope of each features. The first evaluation is meant to identify the related existing features and to prune the actions which are out of scope.

The outcome of the preliminary analysis is an assessment grid (Figure 5.2.1), feeded back to the proposers of the scenarios for validation: to check if considering the mapping between actions and features (components) and the pruning it is still possible from their perspective to provide the service.

		Phase 1	Phase 2	Phase 3	Phase 4
Actor A	Role X	Action 1	Action 3	Action 4	
Actor B		Action 2		Action 5	
Actor C	Role Z		Action 1	Action 3	Action 5

*Figure 5.2.1. General structure of the Assessment Grid, actors are collected accordingly to their roles, at each phase each role may perform different actions.*

The pruning of actions is very critical since it is need to keep the focus of the project within its scope, to keep the resources to develop foundational features (features which are mandatory to provide the services), and to avoid possible backlash coming from controversial features (privacy issues, transparency, usability, high maintenance costs, etc.)

The Assessment Grid is specifically designed to support the assessment of the requests and the use cases reported in the service scenarios, providing the whole pictures of the process (that may involve tens of actors in months), and contextualising the scope and goal of the actions, connecting each action with roles and phases.

From the analysis of the Assessment Grid becomes much easier to extract hidden requirements, for instance related to the “timing” of actions and the cause/effect relations between actions. Moreover, it makes easier to assess the impact of the pruning of actions making easy to expose the interdependencies.

Following the validation, the requirements extracted from the scenarios are mapped to the actions inside a Backlog Grid (Figure 5.2.2). The backlog reports:

- The requirements
- The priority (high or low)
- The status of the development (open, done or rejected)
- The validations flags for all proposers and developers involved
- The related features
- The related components

The Backlog Grid is meant to keep track of the evolution of the development, keeping the overhead low for the development teams and accessible to the municipalities.

Actions	Requirements	Related Features	Proposed Solutions	Component Involved	Validation	Status	Priority
a01	Req. 1 Req. 2 Req. 3	Sigup	Signup page	Componet1	Municipality, Component1	Open	HP
a02	Req. 1	Upload	none	none	Municipality1	Rejected	-

*Figure 5.2.2. General structure of the Backlog Grid: for each action in the Assessment Grid the backlog reports the requirements, the related features (existing or future), the involved components, the validation flags (proposers and developers), the status, and the priority.*

The Backlog Grid provide the abstraction the requirements extracted from the service scenarios, under the light of the features of components.

Some requirements are not strictly related to a process. A Backlog Grid for general requirements is used to collect the “adoption” requirement: the requirements for the general adoption of the platform from the local stakeholders.[Appendix 5.2 - Assessment Grid](#)

The development of the platform should follow the analysis of the outcome of the requirement elicitation and of the scenarios developed by the municipality involved in WeGovNow. The service scenarios provide contextual information about the service the municipalities wish to provide, and use cases grounded on their understanding of the features of the software components, functional and nonfunctional required are extracted and normalised on the bases of the use cases as mentioned in the service scenarios.

The extraction of the development cannot be done from the output of the engagement activities but requires two preliminary steps:

1. **Analysis and clustering of the service scenarios**, each scenario is analysed enquiring the proposers in order to design the general process behind the scenarios. The general process is used as framework to represent the scenario in terms of actors (roles), phases and actions (features).
2. **Assessment and mapping of the actions**, first the actions are evaluated in terms of existing features, scope of the projects and scope of each features. The first evaluation is meant to identify the related existing features and to prune the actions which are out of scope.

The outcome of the preliminary analysis is an assessment grid (Figure 5.2.1), feeded back to the proposers of the scenarios for validation: to check if considering the mapping between actions and features (components) and the pruning it is still possible from their perspective to provide the service.

		Phase 1	Phase 2	Phase 3	Phase 4
Actor A	Role X	Action 1	Action 3	Action 4	
Actor B		Action 2		Action 5	
Actor C	Role Z		Action 1	Action 3	Action 5

*Figure 5.2.1. General structure of the Assessment Grid, actors are collected accordingly to their roles, at each phase each role may perform different actions.*

The pruning of actions is very critical since it is need to keep the focus of the project within its scope, to keep the resources to develop foundational features (features which are mandatory to provide the services), and to avoid possible backlash coming from controversial features (privacy issues, transparency, usability, high maintenance costs, etc.)

The Assessment Grid is specifically designed to support the assessment of the requests and the use cases reported in the service scenarios, providing the whole pictures of the process (that may involve tens of actors in months), and contextualising the scope and goal of the actions, connecting each action with roles and phases.

From the analysis of the Assessment Grid becomes much easier to extract hidden requirements, for instance related to the “timing” of actions and the cause/effect relations between actions. Moreover, it makes easier to assess the impact of the pruning of actions making easy to expose the interdependencies.

Following the validation, the requirements extracted from the scenarios are mapped to the actions inside a Backlog Grid (Figure 5.2.2). The backlog reports:

- The requirements
- The priority (high or low)
- The status of the development (open, done or rejected)
- The validations flags for all proposers and developers involved
- The related features
- The related components

The Backlog Grid is meant to keep track of the evolution of the development, keeping the overhead low for the development teams and accessible to the municipalities.

Actions	Requirements	Related Features	Proposed Solutions	Component Involved	Validation	Status	Priority
a01	Req. 1 Req. 2 Req. 3	Sigup	Signup page	Componet1	Municipality, Component1	Open	HP
a02	Req. 1	Upload	none	none	Municipality1	Rejected	-

*Figure 5.2.2. General structure of the Backlog Grid: for each action in the Assessment Grid the backlog reports the requirements, the related features (existing or future), the involved components, the validation flags (proposers and developers), the status, and the priority.*

The Backlog Grid provide the abstraction the requirements extracted from the service scenarios, under the light of the features of components.

Some requirements are not strictly related to a process. A Backlog Grid for general requirements is used to collect the “adoption” requirement: the requirements for the general adoption of the platform from the local stakeholders.

## **Appendix 5.3**

Example of an assessment grid  
for a service scenario

[illegible]

Citizens (Example: elderly, children, severely impaired, unemployed people, etc.)	E02_Intercity people					C24_Consultation of information about local services and initiatives C25_Request or reservation of services		D17_Participation in meetings and online forums D23_Creation of a group of interest to develop a proposal		E23_Request for services or help (through the reporting system) E30_Sharing of services with peers	
	E03_Local residents			B16_Reporting of critical situations concerning elders and frail people or community issues to the municipal offices	B16:	C26_Publication of competences/skills/free time available for volunteering initiatives				E24_Reporting system about social issues and alerts E25_Organization of open groups at neighborhood level	
	E04_Far owners									E26_Activation of co-housing services and options	
	E05_young or unemployed people					C24_Consultation of information about local services and initiatives				E27_Search for formal and informal training or work opportunities E28_Self-entrepreneurship - Design and experimentation of new economic activities on local needs	

## **Appendix 5.4**

### Example of a backlog grid

Actions	Requirements	Related Features	Proposed solutions	Components involved	Validation	Status	Priority
Identification code of the action (A01, A02, A03, An) + description	Identification code of the requirement (RQA01, RQA02, RQA03, RQAn) + description	Identification code of the feature (FT1, FT2, FT3, FTn) + description	Identification of proponent for alternative solutions + description	Identification code of the component involved:  • FirstLife: FL • LiquidFeedback: LF • ImproveMyCity:IMC • Geokey:GKCommunity maps: CM • Ontomap:OM • Cross-component:CC (id components involved)	Identification code of the status of validation process  \$WP2 + identification code of trial sites \$WP2_TO(Torino)\$WP2_SD(San Donà di Piave)\$WP2_LN (Southwark, London) #WP3 + identification code of the component involved#WP3_FL, #WP3_LF, #WP3_IMC, #WP3_GK, #WP3_OM	State of development of the feature  • Done: the feature already exists in the component • In progress: the development or the adaptation of the feature is on-going • Open: accepted, but to be scheduled • Rejected: the feature requested is incompatible with the setting of components, budget or technological constraints, or the project goals	Evaluation of the priority  • HP: high priorityThe feature is general, requested by all the trial sites, and needed to test the prototype in the service scenario. •LP: low priorityThe feature is not requested by all the trial sites and there are alternative features that can be used to test the prototype in the service scenario
A01_standardization of existing datasets including demographical, epidemiological, economic and social information on a geographical basis for the import on the platform	RQA01a_Tool to speed up the standardization of heterogenous data RQA01b_Format to be imported compatible with the format of existing files and currently used by the Municipality officers (CSV o GeoJason)	FT01_undefined/out of scope		None	\$WP2_SD	Rejected	LP
A02_Automatic import of existing datasets on the platform	RQA02a_The import keep metadata: categories, descriptions, etc. RQA02b_The import can be done periodically by municipal officers (different time intervals for different type od data) RQA02c_The import can be automatized connecting the platform to the existing digital archives of the municipality RQA02d_Automatic creation of a project from a dataset	FT02_Importer		GK? Ontomap?	\$WP2_SD	Open	HP? - to be verified if it could be a shared action, LN - manual importer?
A03_Visualization of each dataset on the map, aggregated at different scales (e.g.: neighborhoods or city)	RQ03a_Data visualization optimized to be comprehensible for not expert users (icons, colours, lists) RQ03b_Selection of data associated to a specific area RQ03c_Filtering of information based on categories, properties, or periods	FT03_Map Viewer	FL: geographical indexing system aggregating spatial units and information	#FL GK?	\$WP2_SD \$WP2_TO	In progress	HP - to be verified with LN
A04_Visualization of multiple connected datasets	RQA04a_Consultation of distinct datasets at the same time RQA04b_Selection of information associated to the same area but coming from distinct datasets (e.g. demographic data + economic data in one of the suburbs) RQA04c_Selection of relevant categories or properties to overlap RQA04d_Thematization of the map overlapping multiple layers	FT04_Filtering System		GK? CM?	\$WP2_SD	Open	
A05_Editing or updating of data	RQA05a_Editing data: text, categories, etc. RQA05b_Editing of areas: changes in geometries or area attributions RQA05c_Identification of the author of edited contents RQA05d_Versioning of data	FT05_Dataset editor	FL: signature for authors belonging to the Municipality	FL? GK? CM?	\$WP2_SD	Open	
A6	RQ6a_						

A7	RQ7a_						
A8	RQ8a_						
A9	RQ9a_						
A10	RQ10a_						
B01	RQ11a_						

## **Appendix 7.1**

### Spatial indexing

# Work report on “pgLatLon”, an alternative to PostGIS

Jan Behrens, Andreas Nitsche

2016-08-19

© 2016 FlexiGuided GmbH, Berlin

## 1 Previous work

As explained in our work report dated June 6, 2016, one created extension for the LiquidFeedback backend (“LiquidFeedback Core”) was the feature of geo-tagging user input as well as allowing geographic searches (geospatial indexing). A common solution for geospatial data processing with PostgreSQL is PostGIS (see <http://postgis.net/>). Using PostGIS to fulfill that task, however, was ruled out because of the following reasons:

- PostGIS would introduce a long chain of software dependencies.
- Many of PostGIS’ features were unnecessary for LiquidFeedback’s needs. At the same time, the number of functions that actually support the WGS-84 spheroid is very limited in PostGIS.
- Viral/incompatible licensing.

Therefore, a lightweight alternative for database indexing and distance measurement (based on a 2-dimensional Taylor series approximation<sup>1</sup> and PostgreSQL’s existing geometric data types<sup>2</sup>) was implemented. The associated results have been published on May 29, 2016 under the terms of the MIT-License.<sup>3</sup>

---

<sup>1</sup>See page 11 of work report June 6, 2016.

<sup>2</sup>See pages 11 through 13 of work report June 6, 2016.

<sup>3</sup>See subsection 2.2.13 of work report June 6, 2016.

## 1.1 Limitations of the implementation as of June 6, 2016

1. While the implementation as of June 6, 2016 was sufficient for local geospatial data processing, it wasn't capable of performing correct calculations for large-area applications (e.g. spanning multiple continents).<sup>4</sup>
2. A particular side-effect of the Taylor-based approach were reversal-symmetry anomalies that could confuse the user of a system when comparing different views on the same dataset.<sup>5</sup>
3. Another drawback was the necessity to include a certain syntax in all geospatial queries (see lines 318 through 339, lines 524 through 544, and lines 606 through 625 in the patched `core.sql` file<sup>6</sup>).
4. Nearest neighbor searches could only be performed iteratively.<sup>7</sup>

As we will show in this document, all previous limitations could meanwhile be lifted by implementing a GiST-based index using fractal curves.

## 1.2 Ideas presented in the work report June 6, 2016

The work report of June 6, 2016 already contained an outlook in regard to possible improvements on page 20:

Keeping future applications in mind, it would be beneficial to improve the implemented methods for geospatial indexing and radial searches by providing an alternative formula for distance calculation on the WGS-84 ellipsoid that could serve as a compromise between the complexity of Vincenty's formulae and the simplicity of local Taylor approximation. Utilizing the nearest neighbor search capabilities of PostgreSQLs GiST indexing framework would be another improvement.

---

<sup>4</sup>Examples for the accuracy have been given in subsection 2.2.8 of work report June 6, 2016 and will be revisited in this work report, subsection 2.1.4

<sup>5</sup>See subsection 2.2.9 of work report June 6, 2016.

<sup>6</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/file/3e28fd842354/core.sql](http://www.public-software-group.org/mercurial/liquid_feedback_core/file/3e28fd842354/core.sql)

<sup>7</sup>See subsection 2.2.11 of work report June 6, 2016.

## 2 Implemented improvements

### 2.1 A new formula for approximating distances between two points on the WGS-84 spheroid

The idea of developing a new algorithm for distance calculation has been pursued and successfully implemented.

#### 2.1.1 The general idea

The general idea to quickly calculate distances on the WGS-84 spheroid is to calculate the exact coordinates of two points in a three dimensional (Euclidean) coordinate system first. Then, the exact tunnel distance can be calculated using the Pythagorean theorem. After the tunnel distance has been determined, a sphere model of earth (with radius  $r = \frac{2a+b}{3}$ ) is used<sup>8</sup> to transform a tunnel distance into a distance on the surface of earth.

$$s_{\text{surface}} = 2r \cdot \arcsin\left(\frac{s_{\text{tunnel}}}{2r}\right)$$

The error for small distances tends towards zero. (Note that for  $s_{\text{tunnel}} = 0$ , the derivative of the above function is 1, and therefore  $s_{\text{surface}} \approx s_{\text{tunnel}}$ .) For medium distances, the above formula serves as a good approximation for the surface distance on the WGS-84 spheroid. In case of antipodal points, however, this method becomes numerically unstable,<sup>9</sup> which is why some modifications are necessary as explained in the following subsection 2.1.2.

#### 2.1.2 Using antipodal points for huge distances

For nearly antipodal points, the distance between two points  $\mathfrak{A}$  and  $\mathfrak{B}$  can be approximated by calculating a third point  $\overline{\mathfrak{B}}$  on earth by reflecting  $\mathfrak{B}$  through the earth’s center. Then the method described in subsection 2.1.1 can be used to calculate a distance  $\overline{s}_{\text{surface}}$  between  $\mathfrak{A}$  and  $\overline{\mathfrak{B}}$  (i.e. between  $\mathfrak{A}$  and the reflection of  $\mathfrak{B}$ ). The approximate distance between the original points  $\mathfrak{A}$  and  $\mathfrak{B}$  then calculates as  $s_{\text{surface}} = \pi r - \overline{s}_{\text{surface}}$ .

Cross-fading is done for points that are neither close to another nor nearly antipodal. This ensures monotonic behavior of the algorithm. For details, we refer to the corresponding source code that has meanwhile been published by the Public Software Group e. V.

<sup>8</sup>With  $a$  and  $b$  being the semi-major and semi-minor axis of the WGS-84 reference spheroid.

<sup>9</sup>There is a numerical instability because the tunnel distance is almost constant for all nearly antipodal points on a sphere. Furthermore, the tunnel distance between two antipodal points on the spheroid varies and is not always  $\leq 2r$ .

### 2.1.3 Costs of trigonometric calculations

The algorithm described in subsections 2.1.1 and 2.1.2 requires 12 trigonometric operations, 5 square root operations, and up to six if-clauses<sup>10</sup> (see source code). For points that are close to one another, some of these operations can be omitted.

Tests have shown that the previously used Taylor-based approach doesn't have any significant speed advantages when implementing both functions in the programming language “C”. The Taylor-based approach still has advantages when trigonometric functions are called in an interpreted language such as PostgreSQL's built-in PL/pgSQL language.<sup>11</sup> By switching to C, however, this advantage vanishes if the pre-calculated Taylor coefficients must be read from a PostgreSQL record-type data structure.<sup>12</sup>

The speed of the new distance calculation function is approximately the same as the old Taylor-based formula while the new formula has a higher accuracy and also works across continents and across the poles.

### 2.1.4 Comparing the accuracy with the truncated Taylor series

The work report from June 6, 2016 presented exemplary errors when calculating distances between the following locations on earth:

Location	Sym.	Latitude	Longitude
Berlin (Brandenburg Gate)	B	52.5162746 N	13.3777040 E
London (Big Ben)	L	51.5007292 N	0.1246254 W
Paris (Eiffel Tower)	P	48.8583701 N	2.2944813 W
Rome (Colosseum)	R	41.8902102 N	12.4922309 W
Moscow (Red Square, Mausoleum)	M	55.7537117 N	37.6198846 E
Longyearbyen (harbour)	X	78.2289157 N	15.5994530 W

The Taylor-based approach resulted in the following relative errors:

<sup>10</sup>Counting the fabs() function as if-clause.

<sup>11</sup>Due to the overhead of function calls.

<sup>12</sup>The PostgreSQL source code contains a comment in backend/executor/execQual.c explaining that the C functions “GetAttributeByName” and “GetAttributeByNum” are slow due to a “typcache” lookup on each call.

Error in ‰	To B	To L	To P	To R	To M	To X
From B (Berlin)	0	+1.40	+1.41	+1.96	+4.42	−6.95
From L (London)	+1.39	0	+0.15	+0.08	+11.09	−3.97
From P (Paris)	+1.29	−0.21	0	+0.09	+10.72	−3.94
From R (Rome)	+0.63	−1.23	−0.66	0	+10.58	−3.03
From M (Moscow)	+4.48	+11.20	+11.15	+13.26	0	−2.01
From X (Longy.)	+1.82	+1.34	+1.55	+2.25	+11.65	0

The new antipodal cross-fading method (as described in subsections 2.1.1 and 2.1.2) results in the following reduced errors:

Error in ‰ <sup>13</sup>	To B	To L	To P	To R	To M	To X
From B (Berlin)	0	0.006	0.008	0.022	0.017	0.056
From L (London)	0.006	0	< 0.001	0.005	0.042	0.051
From P (Paris)	0.008	< 0.001	0	0.003	0.050	0.056
From R (Rome)	0.022	0.005	0.003	0	0.084	0.065
From M (Moscow)	0.017	0.042	0.050	0.084	0	0.070
From X (Longy.)	0.056	0.051	0.056	0.065	0.070	0

While there is no formal proof, tests have shown that the maximum error of the new method is always less than 2 ‰ (0.2 %) for random points on earth.<sup>14</sup> Therefore, the first limitation mentioned in section 1.1 gets lifted when using the new formula for calculating distances between points on earth.

### 2.1.5 Reversal symmetry

Also the second limitation mentioned in section 1.1 gets lifted by using the new formula: as already visible in the last table of the previous subsection 2.1.4, the error of the new distance function is not only smaller but also symmetric (e.g. the error when calculating the distance from London to Moscow is the same error as calculating the distance from Moscow to London). This is an improvement when comparing the method to the Taylor-based approach, which doesn't behave symmetrically (see corresponding tables in the previous subsection).

To ensure that not even floating point errors can yield to different results when swapping the origin and destination point, our implementation orders the latitude and longitude values prior to calculation (see source code for details).

<sup>13</sup>All errors in this table have a positive sign.

<sup>14</sup>This is less than the maximum error when earth would be completely modeled as a sphere.

## 2.2 Utilizing PostgreSQL’s Generalized Search Tree (GiST) framework

While the improvements described in section 2.1 can lift the first and second limitation mentioned in section 1.1, a mere formula for calculating distances cannot improve the syntax constraints (third issue) or allow nearest neighbor searches without the incremental workaround (fourth issue in section 1.1).

For these reasons, we considered using the GiST interface of PostgreSQL to provide an integrated solution for geospatial indexing.<sup>15</sup>

### 2.2.1 A short introduction to GiST

For the remainder of this section 2.2, we assert that the reader be familiar with the general structure of GiST indices. A short introduction to GiST can be found at the following URLs:

- <http://gist.cs.berkeley.edu/>
- <http://db.cs.berkeley.edu/papers/vldb95-gist.pdf>
- <https://www.postgresql.org/docs/9.5/static/gist-intro.html>

An implementation of a nearest neighbor search capable GiST index in PostgreSQL requires the implementation of the following 8 support functions: “consistent”, “union”, “compress”, “decompress”, “penalty”, “picksplit”, “same”, “distance”.<sup>16</sup> The structure of the index differs depending on the particular implementations chosen for these functions.

After discussing the key structure for keys stored in the index (based on a space-filling fractal curve) in subsection 2.2.2, a short overview on the particular implementation of each of the 8 GiST support functions will be given in subsection 2.2.3.

### 2.2.2 Using a space-filling curve

While both PostgreSQL and PostGIS use GiST-based R-tree indices for geometric (and, in case of PostGIS, geographical) data types, an R-tree based implementation didn’t appear to be the best choice for geospatial indexing.<sup>17</sup> Instead, we

<sup>15</sup>This was already proposed in section 3.3 (“Remaining tasks”) of work report June 6, 2016 as well as footnote 45 on page 16 of the same document.

<sup>16</sup><https://www.postgresql.org/docs/9.5/static/gist-extensibility.html>

<sup>17</sup>An R-tree usually has overlapping nodes and requires to store bounding boxes for each node. For these two reasons, additional working memory would be consumed which, in turn, can be a problem for caching indices on huge datasets in main memory.

used an index based on a space-filling fractal curve. For the fractal, the Lebesgue curve<sup>18</sup> was chosen for reasons of implementational simplicity.

Following the shape of a Lebesgue curve, a single point on earth can be represented by a bit string where the latitude and longitude bits are interspersed:

$$\phi_0, \lambda_0, \phi_1, \lambda_1, \phi_2, \lambda_2, \phi_3, \dots, \phi_{27}, \lambda_{27}$$

with

$$\phi_n \in \{0, 1\}, \lambda_n \in \{0, 1\}$$

This representation consumes 56 bits including the zeroth bits of latitude and longitude because  $2 \cdot (27 + 1) = 56$ . The worst resolution of the index is  $40,000 \text{ km} / 2^{28} \approx 14.9 \text{ cm}$  at the equator.<sup>19</sup>

The leaves of the tree always contain 56 bit-long bit strings (7 bytes). The internal nodes of the search tree, however, need to cover a range of leaves. For this reason, truncated bit strings need to be stored in the nodes. This is encoded by appending a length information (requires 6 bits to store a length from 0 to 56 inclusive) to the key.

In practice, the total key size for single points is rounded up to 8 bytes for reasons of memory alignment and polymorphic functions (refer to the source code for details).

Beside storing simple points, the index shall also be capable of handling more complex objects that cover an area bigger than a singular point (e.g. paths, polygons, etc). To store such kinds of objects (and to allow index lookups on those entries), a center point of a bounding circle and its radius is included in the index keys. The center point can be encoded by interspersing the latitude and longitude bits as explained above. However, the radius of the bounding circle (i.e. the information on the size of the indexed object) must also be interspersed with the bits for latitude and longitude of the center of the bounding circle. The following scheme can be used:

$$\rho_0, \phi_0, \rho_1, \lambda_0, \rho_2, \phi_1, \rho_3, \lambda_1, \rho_4, \phi_2, \rho_5, \dots, \rho_{54}, \phi_{27}, \rho_{55}, \lambda_{27}, \rho_{56}$$

As the exact radius of the bounding circle doesn't need to be stored precisely,<sup>20</sup> we can restrict the choice of all  $\rho_n$  in such way that only one bit can be

<sup>18</sup>[https://en.wikipedia.org/wiki/Z-order\\_curve](https://en.wikipedia.org/wiki/Z-order_curve)

<sup>19</sup>Note that the resolution of the index is not limiting the resolution of the indexed data because PostgreSQL can recheck whether a condition is satisfied using the table data rather than the index data.

<sup>20</sup>A larger value for the radius can be stored because PostgreSQL can recheck whether a condition is satisfied when querying the index. See “consistent” support function on page 8 and the associated footnote 24.

set to 1 while all other bits are 0. This allows to perform a logarithmic compression of  $\rho$  by simply storing an index  $n$  of that  $\rho_n$  which is 1.<sup>21</sup> If  $n = 57$ , then all  $\rho_n = 0$ . Since  $n \in \{0, 1, 2, \dots, 57\}$ , appending a single byte to the index key is sufficient to consider the dimensions of an object referenced in the index tree. The total key size results in 9 bytes when indexing geographic objects with a size not always equal to zero (i.e. circles, paths, polygons, etc).

Object type	Key size of leaf nodes <sup>22</sup>	Key size of internal nodes
Points	7 bytes	8 bytes
Boxes	8 bytes	9 bytes
Circles	8 bytes	9 bytes
Paths	8 bytes	9 bytes
Polygons	8 bytes	9 bytes

Using the space-filling fractal as well as the logarithmic compression results in a maximum key size of 9 bytes, which is considerably smaller than storing the position and dimension of bounding boxes as done in R-trees.

### 2.2.3 Implementation of the 8 support functions

The following list gives an overview on the implementation of the necessary GiST support functions. For details, please refer to the source code.

**consistent** PostgreSQL allows three return values: “no”<sup>23</sup>, “maybe”<sup>24</sup>, or “yes”<sup>25</sup>. Because the index key format (as explained in subsection 2.2.2) is lossy<sup>26</sup>, only “no” and “maybe” are returned while “yes” is never returned.<sup>27</sup>

For equality queries, the query datum is simply converted to a key, and then “maybe” is returned if and only if the key of the query datum is equal to the key stored in the tree.

<sup>21</sup>Increasing  $n$  by 1 corresponds to reducing the area of the bounding circle by 50 %, hence dividing the radius by  $\sqrt{2}$ . The special value of  $n = 0$  is used for all bounding circles with a radius greater than a given reference length (which is in the same order of magnitude as the radius of earth).

<sup>22</sup>PostgreSQL does not allow to specify different key sizes for leaf nodes and internal nodes unless using variable size data types. To reduce computational and implementational overhead, leaf nodes use the format for internal nodes. Hence the values in this column are not applicable for the implementation.

<sup>23</sup>Returning false.

<sup>24</sup>Returning true with “recheck” set to true.

<sup>25</sup>Returning true with “recheck” set to false.

<sup>26</sup>The resolution can be as coarse as 14.9 cm at the equator.

<sup>27</sup>Note that while it still would be possible to return “yes” in certain cases, the implementational and computational overhead doesn’t seem to justify the necessary extra calculations.

For testing whether a point is contained in a bounding box, the key stored in the tree is converted to a bounding box, and then “maybe” is returned when this bounding box overlaps with the query bounding box.

For all other “overlap” queries,<sup>28</sup> a minimal distance between the query object (e.g. a circle for radial searches) and a bounding box<sup>29</sup> for the center point of the bounding circle of the indexed object is calculated. The maximum size of the indexed object (which is stored in the key using logarithmic compression) is subtracted from that value. If the value is smaller than or equal to zero, “maybe” is returned.

In all other cases, “no” is returned.

**union** Combining two keys is performed by simply truncating the bit strings of the keys (at the right side) to have the same length and then further truncating any differing bits from the right side of the strings.

**compress** The compress function converts geographic objects to the index key structure as defined in subsection 2.2.2.

**decompress** The decompress function is not used (and simply returns its argument) because the index support functions work directly on the stored key data format.

**penalty** The penalty function returns the number of truncated bits when creating a union of an original key with a key to be inserted. This roughly corresponds to returning the additionally covered area in case of R-trees.

**picksplit** The picksplit function decides which entries are moved to a new page when the size of the page grows too big, and which entries are kept. In case of R-trees, it is possible that entries are both kept and copied to a new page (overlapping bounding boxes). The fractal curve described in subsection 2.2.2 does not require overlapping index pages though. Instead, a union of all keys in the page is calculated. Then one bit is appended to the bit string of the union (this bit is set to 0 or 1). Those keys that are covered by one key (e.g. the union appended by 0) stay on the old page, and those keys that are covered by the other key (e.g. the union appended by 1) are moved to the new page. If the bit string of the union of all keys has already a maximum length (i.e. if no bit can be appended to the bit string), then a trivial split is performed where half of the entries are selected arbitrarily to be moved to a new page.

---

<sup>28</sup>Operator “&&” in SQL.

<sup>29</sup>Estimating the possible location of the center point, calculated from the (lossy) index key.

**same** The “same” support function simply tests two keys for equality. In our implementation, this check can be performed binary (using memcmp) because all unused bits are always initialized.

**distance** The distance function returns a minimal distance between the query object and the bounding box<sup>29</sup> of the center point of the bounding circle of the indexed object. Then the maximum size of the indexed object (which is stored using logarithmic compression) is subtracted. (This is basically the same mechanism described for the “consistent” function.)

The result is sanitized in such way that only zero or positive finite values are returned because PostgreSQL internally uses  $-\infty$  and  $+\infty$  for other purposes, such as handling NULL values.<sup>30</sup>

## 2.2.4 Handling empty objects

Special considerations have been taken to handle empty objects in the index (objects which do not contain any point and thus do not overlap with any other geographic object or even themselves). These could be circles with a negative radius or a set of polygons that is empty.

In order to process empty objects (which are distinct from NULL), the “compress” function must return a special key (the “empty key”) that is not matching any location on earth. Furthermore, another special key (the “universal key”) covering both empty and non-empty geographic objects must be returned by the “union” function when combining keys of empty and non-empty objects.

The two special keys “empty key” and “universal key” are implemented without additional space consumption by storing a magic value in the byte that is normally storing the logarithmically compressed object size. For the details of the implementation, refer to the source code.

## 2.3 Adding new data types and necessary operators

In addition to providing the previously explained key format and associated GiST support functions, actual implementations for the desired operators had to be created. Most importantly, that is:

- the overlap operator ( $\&\&$ ),
- the distance operator ( $\<->$ ).

For most other operators such as the “contains” ( $\&\>$ )<sup>31</sup> and the “contained in”

<sup>30</sup><https://www.postgresql.org/docs/9.5/static/gist-extensibility.html> warns about returning  $\pm\infty$ .

<sup>31</sup>In older PostgreSQL versions:  $\sim$  (still used by PostGIS).

(<@)<sup>32</sup> operator, the same index strategy as used for the overlap operator (&&) can be applied.<sup>33</sup> Only the operator functions (and necessary geometric operations) would need to be implemented for adding these operators.

The following data types have been implemented:

- `epoint`: a point on earth,
- `ebox`: a latitude/longitude range (e.g. describing a viewport for a map),
- `ecircle`: a filled circle (used in conjunction with the distance operator for radial searches with fixed radius),
- `eclass`: a collection of points, paths, (filled) polygons, and outlines (i.e. closed paths / non-filled polygons).

Particularly for polygons, certain geometric algorithms had to be implemented such as a “point in polygon” algorithm based on ray tracing<sup>34</sup> or geometric projections to determine the shortest distance between two polygons or a point and a polygon. As a result, the distance operator <-> works with high precision and does not use bounding boxes for an approximation of the distance to clusters/polygons. (Note: PostGIS, in contrast, uses inaccurate bounding boxes here.)

Currently not all combinations of operators and/or data types are implemented. This is ongoing work; for details refer to the published reference documentation.

## 2.4 Treatment of the poles and the 180<sup>th</sup> meridian

The distance calculation algorithm as explained in section 2.1 works properly both at the poles and the 180<sup>th</sup> meridian. The new algorithm automatically considers shortest paths crossing the north or south pole when calculating distances (e.g. between Iceland and Alaska). No special handling is required here.

The fractal index as explained in section 2.2 has a slight drop of performance near the poles because of the singularity in the coordinate system (at the poles all possible longitudes from -180 to +180 degrees are covered). The accuracy, however, is not affected by the index.

Whenever clusters are used (which may contain paths, polygons, or outlines of polygons), there is a drop in accuracy next to the poles that depends on

<sup>32</sup>In older PostgreSQL versions: @ (still used by PostGIS).

<sup>33</sup>If an object is contained in another object, those two objects always overlap.

<sup>34</sup>[https://en.wikipedia.org/wiki/Point\\_in\\_polygon](https://en.wikipedia.org/wiki/Point_in_polygon)

the maximum distance between the vertices of the polygon. This is because projections are performed in Euclidean space.

In order to avoid ambiguities in regard to eastern/western orientation of edges of paths, polygons, or outlines of polygons, each entry in a cluster may only cover a longitude range of less than 180 degrees. Areas that cover a wider longitude range are still possible by splitting them into multiple polygons.

### 3 Comparison with PostGIS

As PostGIS is only available under the terms of a viral license that is incompatible with certain other open source licenses,<sup>35</sup> using PostGIS is not an option for the LiquidFeedback project.<sup>36</sup> However, we will compare our created solution with PostGIS for informational reasons.

#### 3.1 Constraints in PostGIS when treating the earth as a spheroid

PostGIS does not support global coordinates for a wide range of functions. To name a few examples:

- The `ST_Overlaps` function is not defined for the geographic data type.
- The `<->` operator returns true distances only between values of the geometry data type (which doesn't work globally) but uses “sphere distance” for the geography data type (which can describe global coordinates).
- The `&&` operator uses bounding boxes for determining whether two objects overlap (i.e. the operator may return true even if two objects do not overlap). This is an even bigger error than just modeling earth as a sphere.

Our created extension, in contrast, always uses spheroid coordinates for all operations.<sup>37</sup> Distances are measured in such way that all local operations honor the flattening of earth, even if the data type and/or table column stores global coordinates. Only for long-range distances, the error can grow up to 0.2 %, which is still better than modeling earth as a sphere.

---

<sup>35</sup>Refer to page 9 of work report June 6, 2016.

<sup>36</sup>LiquidFeedback aims to provide users and developers maximum freedom and accessibility to the source code also when combining it with any other software components. The GNU General Public License is contrary to these goals. See also: <http://www.public-software-group.org/licenses>

<sup>37</sup>The only exception is when internally performing projections on a polygon, path, or outline in order to determine the shortest distance to a cluster (see section 2.4). The error, however, approaches zero in most practical cases.

### 3.2 OpenGIS “Simple Features Specification for SQL”

PostGIS provides many functions such as `ST.Distance`, `ST.MakePoint`, etc. as defined in standards by the OpenGIS consortium as well as ISO 19125. Implementing these functions as well as data storage formats that follow the OpenGIS geometry class hierarchy is possible but a matter of time and resources (see also following section 4.2 on compatibility considerations regarding additional data storage formats). It should be noted that PostGIS does *not* properly implement many of the functions when using spheroid coordinates that work across several continents, i.e. globally. This has already been elaborated in the previous subsection 3.1.

## 4 Compatibility considerations

### 4.1 Standard compliance

GIS standards may be followed on the SQL level or on the level of data representation when querying a database through REST API calls.

As already mentioned in section 3.2, extending our geospatial extension for PostgreSQL to include functions such as `ST.Distance`, `ST.MakePoint`, and associated data types on the SQL level is possible but a matter of time and resources (and budget).

Beside these economic concerns, it is currently not planned in the WeGovNow consortium to provide any form of SQL interface to third party components. Instead, a REST based API has been proposed (and is already implemented by GeoKey, for example). Therefore, compatibility must be ensured on the level of data representation when querying WeGovNow software components through REST API calls.

As we will see in the following section 4.2 on additional data storage formats, PostgreSQL enables a programmer to include any desirable data format with little overhead. Therefore, it is easily possible to make a REST API interface standard compliant with any kind of geospatial data exchange format including those which follow the OpenGIS geometry class hierarchy or any other scheme like GeoJSON (which is used by UCL's GeoKey).

### 4.2 Additional data storage formats

It is desirable to also support more complex geographic data formats such as GeoJSON or XML based documents, which may contain meta data. Whenever

needed, these formats can be stored directly in a corresponding JSONB, XML<sup>38</sup>, or even TEXT table column.<sup>39</sup> All spatial indexing functions and operators as presented in section 2 of this paper, may still be used for these data formats even if the data types do not support lossless round-trip conversions. This is possible because PostgreSQL supports indexes on expressions (see chapter 11.7 of the PostgreSQL 9.5 manual<sup>40</sup>). An application simply needs to provide a one-way mapping function from the stored geographic data type to one of the data types supported by the index.

### 4.3 Using a different spheroid

Also other reference spheroids than WGS-84 are thinkable by adjusting the corresponding constants in the C code. However, as the WeGovNow project depends on a compatible standard for coordinates anyway, it would be advisable to use WGS-84 as a common standard.

## 5 Publication of results

The developed software has been submitted to the Public Software Group e. V on August 18, 2016 and was published by the Public Software Group in the source code repository of LiquidFeedback Core<sup>41</sup> under the terms of the MIT-License<sup>42</sup> on August 18, 2016.

The source code has been modularized so that it is easy for other consortium partners to incorporate it, if desired. All previous work on LiquidFeedback in regard to spatial data was adjusted in order to use the new geospatial indexing module.

---

<sup>38</sup>Support for the XML data type requires PostgreSQL to be compiled with the “--with-libxml” configuration option.

<sup>39</sup>Such data types may be extended by adding validity checks using so-called “domains”. See PostgreSQL’s documentation on the CREATE DOMAIN command: <https://www.postgresql.org/docs/9.5/static/sql-createdomain.html>.

<sup>40</sup><https://www.postgresql.org/docs/9.5/static/indexes-expressional.html>

<sup>41</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/rev/96ee2db56bec](http://www.public-software-group.org/mercurial/liquid_feedback_core/rev/96ee2db56bec)

<sup>42</sup>[http://www.public-software-group.org/mercurial/liquid\\_feedback\\_core/file/96ee2db56bec/LICENSE](http://www.public-software-group.org/mercurial/liquid_feedback_core/file/96ee2db56bec/LICENSE)

## 6 Conclusion & Outlook

It has been shown that PostGIS' spatial indexing functions can be replaced by a more lightweight approach within reasonable time. Previous limitations (see section 1.1) could be lifted. A usable implementation has been published as open source software under a permissive (non-viral) license.

While certain data representation formats are not yet supported by the new extension, these could easily be integrated with the current implementation (refer to section 4 for that matter).<sup>43</sup>

© 2016 FlexiGuided GmbH, Berlin

---

<sup>43</sup>Note, however, that other data structures (such as bézier curves etc.) would require further data type implementations.

## **Appendix 7.2.1**

### Portrayal of quality information

### Appendix 7.2.1. Portrayal of quality information

With regards to the data quality portrayal task of the project, a number of JavaScript libraries are being implemented in a testing environment that allows for the identification of feasibility of various portrayal methods identified from the literature. Such methods include the dynamic colouring of features, giving features a “sketchy” appearance, and enabling transparency of features. The geographic features are drawn on a webmap using the Leaflet JavaScript libraries. Due to the modular nature of the portrayal libraries created, they could be easily imported into existing tools and used to stylise features that contain quality information. This collection of portrayal libraries will be continually added to throughout the development phase of the project as additional methods are evaluated. Currently, 23 different methods for representing information have been identified from the literature. Not all of these methods are applicable to all forms of data however, with methods such as changing the shape of an icon only being applicable to point information. Not all methods have been assessed for suitability and as this is done during the prototyping phase, libraries will be implemented where required.

The tool used for implementing and testing the portrayal methods is a simple HTML webmap using Leaflet technology (Figure 7.2.1.1). The page allows the entering of GeoJSON data through a text area or from direct access to a secondary datasource (such as an API request). Once entered, the user selects an attribute of the data features which represents the quality indicator that is to be portrayed. They then select the portrayal method from a list (dynamically populated based on the libraries present) and the map is updated to show the features with the method selected. Figures 7.2.1.2 and Figure 7.2.1.3 show a portrayal of OSM road data with the timestamp used as the indicator value. The two portrayal methods shown are Sketchy (Figure 7.2.1.2) and colour hue (Figure 7.2.1.3). Though this representation is of OSM data acquired via the Overpass Turbo<sup>1</sup> tool, any geographic information can be portrayed so long as it is in the GeoJSON data format. For example, data obtained through the OnToMap service is delivered in a GeoJSON format and so could be directly represented within the testing tool. However, any data being represented would need to have an attribute representing the quality information that is being portrayed (with the exception of a timestamp field which automatically is represented as older being lower quality/higher uncertainty).

---

<sup>1</sup> <https://overpass-turbo.eu/>

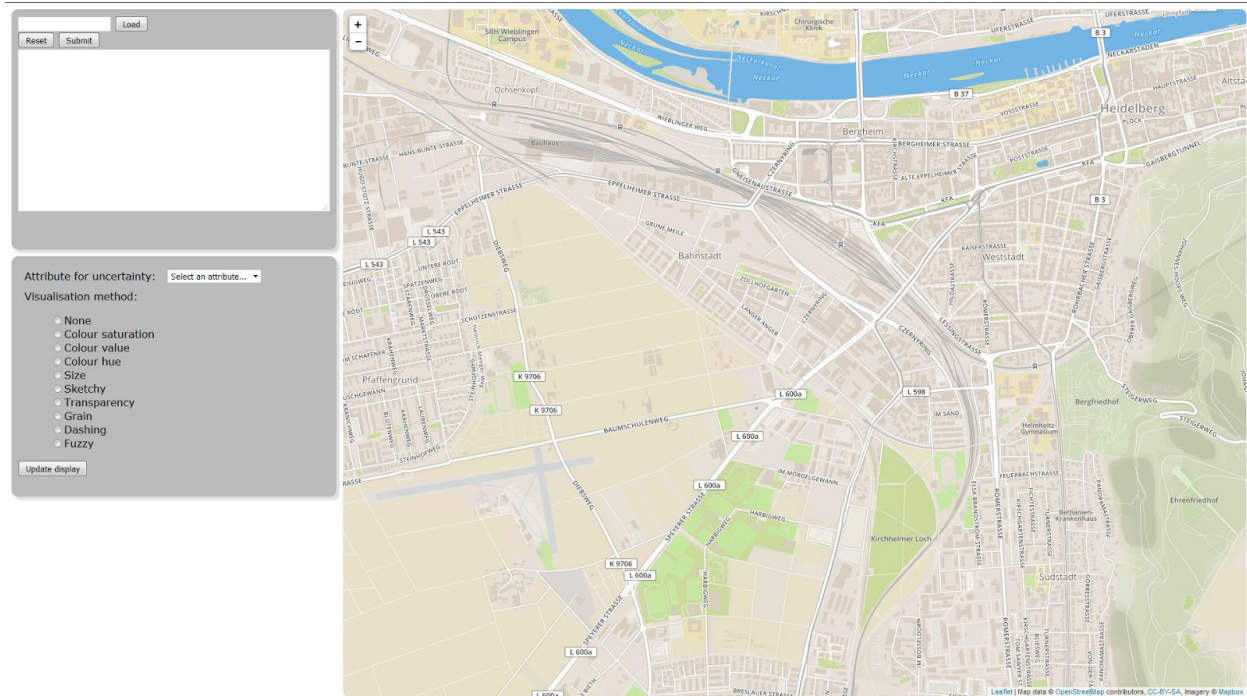
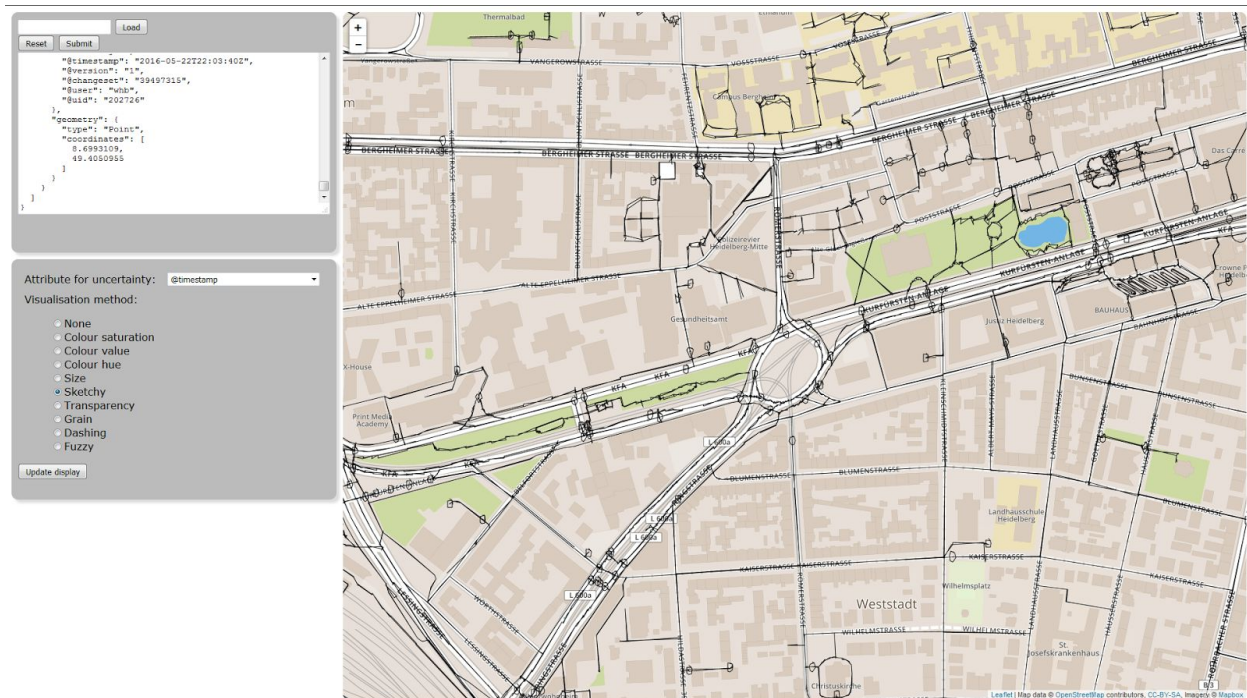
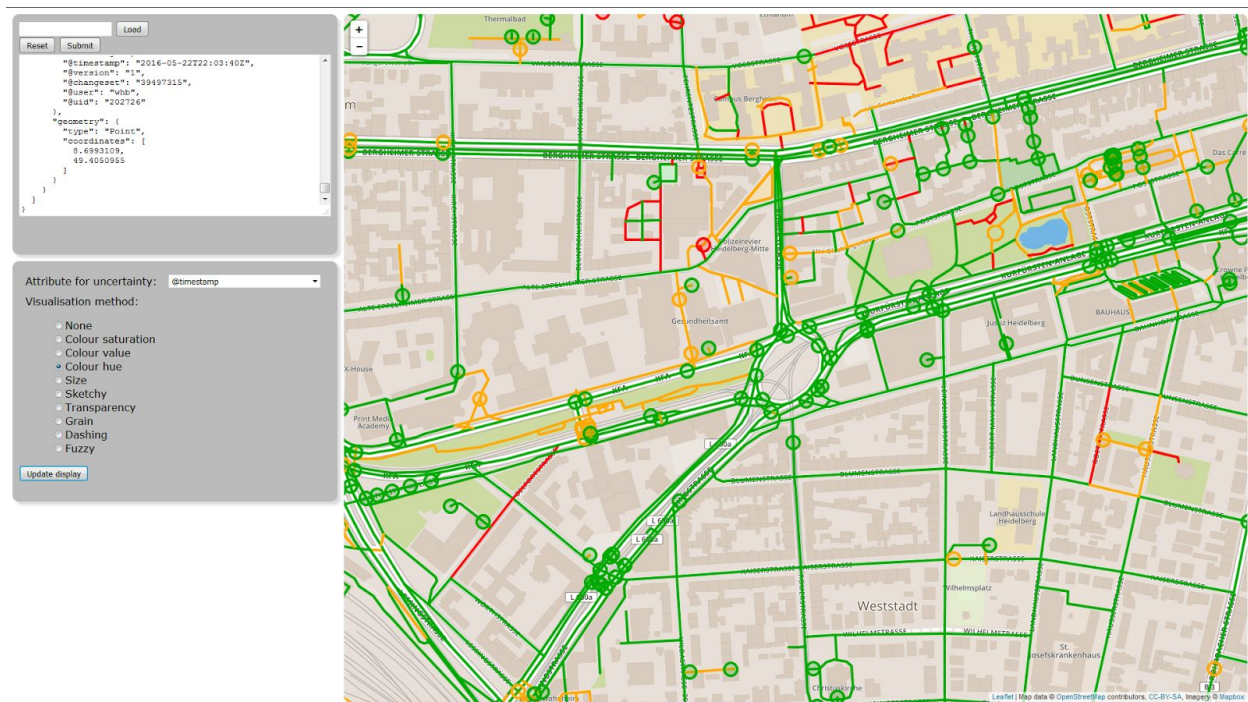


Figure 7.2.1.1: portrayal testing platform (base map © OpenStreetMap contributors)



*Figure 7.2.1.2: Age of road data shown in a sketchy form (more sketchy appearance = older data) (base map © OpenStreetMap contributors)*



*Figure 7.2.1.3: Age of road data shown in a coloured form (old data = red, newer data = green) (base map © OpenStreetMap contributors)*

When reading the GeoJSON data that is to be represented, the tool investigates each feature and stores the attributes present. A list of all attributes is presented in the drop down where the user can select which attribute to use as the portrayal value. In the future, it is envisioned that an automatic value for attribute completeness will be calculated, but this is dependent on methods identified when reviewing the data quality literature.

Portrayal of features, as mentioned, is performed using the Leaflet<sup>2</sup> (v1.0.1) web mapping library. This library was chosen because of its wide use in other services and the ability to directly import GeoJSON data into the library for display on the map. The changing of the display according to the portrayal method selected is based on whether the methods changes geometry or not. For example, methods such as colour hue or transparency which change the appearance of the feature on the map but do not change its geometry are accomplished by changing the style associated with features. Other methods such as the Sketchy rendering technique require a direct manipulation of the geometries. The methods for generating a

<sup>2</sup> <http://leafletjs.com/>

sketchy representation can be found in Wood et al. (2012)<sup>3</sup>. These methods were implemented as closely as possible in the portrayal testing tool. This method along with others such as fuzzy rendering also make use of the Turf<sup>4</sup> (v3.5.1) javascript library. This library allows for the quick and relatively easy implementation of simple GIS techniques such as buffering on features. If a change in geometry is required, the original geometry is stored in a temporary variable and the altered geometry displayed on the map. All attributes associated with the original features are attached to the new geometries. When a new representation is selected, the original geometry is reloaded from the temporary variable and used in any processing.

At this stage, the concept of the testing tool is to allow a relatively easy portrayal of different data which not only allows the development of the portrayal libraries, but also allows users to look at the different methods and choose which they prefer. This is a key aspect for investigating the effectiveness of the methods prior to a live implementation of the WeGovNow platform.

The following table depicts the visual variables and portrayal methods that have been currently identified from the literature for visually portraying information. The methods that have currently been implemented in the portrayal testing tool are marked with implemented=yes and those that require more investigation before implementation are marked as implemented=no. n/a in the implemented column indicates that that particular variable is not suitable for the indicated feature type (i.e. changing an icon is only suitable for point features, and not for lines and polygons).

		Implemented		
Name	Description	Point	Line	Poly
Weight/size	Size of the “pen” used to draw features	yes	yes	yes
transparency	How opaque the features are	yes	yes	yes
fuzzyness	The sharpness of feature boundaries	yes	yes	yes
Colour hue	Colour used to draw features (red, green, blue etc.)	yes	yes	yes
Colour saturation	Intensity of the colour	yes	yes	yes
Colour value	Lightness/darkness of a colour	yes	yes	yes
Grain	The closeness of lines in hatch-filled shapes	yes	n/a	yes
Sketchiness	How much like a quick sketch the features look like	yes	yes	yes

<sup>3</sup> Wood, J., Isenberg, P., Isenberg, T., Dykes, J., Boukhelifa, N. and Slingsby, A., 2012. Sketchy rendering for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), pp.2749-2758.

<sup>4</sup> <http://turfjs.org/>

Dashing	The size of dashing used on lines to draw features	yes	yes	yes
Fog overlay	An overlay of varying opaqueness that obscures features	no	no	no
Icon	A symbolic icon to represent the feature	no	n/a	n/a
Resolution	The size of pixels used to draw features	no	no	no
Location	A displacement of the feature	no	no	no
Shape	The shape used to represent the feature	no	n/a	n/a
Texture type	The type of texture used to fill a feature	no	n/a	no
Texture arrangement	The relation of elements used to generate a texture filling the feature	no	no	no
Colour mixing		no	no	no
Grain orientation	The direction of the hatches used in a hatch-filled feature	no	n/a	no
Separate overlays	Use of multiple overlays to portray information. Separate overlays could contain statistical information	no	no	no
Dynamic features	A continuous alteration of the feature during representation, such as blinking or gradual increase in size	no	no	no
Glyphs/Error bars	Adding a representation to the feature that portrays a number of elements, such as spider diagrams	no	no	no
Grid overlay	Applying an overlay to the data that xxx	no	no	no
Contouring	Using contours to group together features of similar value	no	no	no

## **Appendix 7.2.2**

### **Spatial Data Assessment Tool**

## Appendix 7.2.2 - Spatial Data Assessment Tool

The reviewed algorithms for spatial data quality assessment are described in very different ways. In many cases, it is difficult to estimate suitability of an algorithm. The most relevant approaches should be implemented and applied for pilot sites. Because of this, it was decided to deploy a spatial data assessment tool (SDAT) allowing us to evaluate and understand existing methods better, as well as, to develop new and derived algorithms or indicators.

The developing tool comprises 3 main blocks. First block contains solutions for preparation spatial databases. Second block include scripts and SQL queries for data assessment parameters calculation. Third block presents results prepared on the previous stage.

SDAT is designed for desktop usage. Results of assessment (mainly graphs, maps, statistics) will be included to a data quality report.

OpenStreetMap data are used at the initial stage. Currently two data sources are utilized: OSM full history dump and OSM map tiles access logs. The access logs are provided by OSM planet as compressed plain text files. Each line of file contains tile number in x/y/ZoomLevel format and number of hits in the considering day. In order to convert this data to SQL spatial database, tile-logs-to-sqlite application was developed (the source code is resided in SDAT's src directory (<http://sdat.n-kov.com/src/>). It has command line interface, the usage is as follows:

```
tclsh tile-logs-to-sqlite.tcl path/to/log/dir path/to/new/database.sqlite
"tile1,tile2,...,tileN"
  path/to/log/dir - path to directory with *.txt.xz files. They could be downloaded by
                    this way: wget --no-parent -r
http://planet.openstreetmap.org/tile_logs/
  path/to/new/database.sqlite - path to output sqlite file. The file must not exist!
  "tile1,tile2,...,tileN" - tiles of interes. Only tile with zoom level greater than
                           zoom level of specifiyed tiles and insile bbox of specified
                           tiles will be considered and saved into output database.
                           Tiles are descrided in format "zoom_level/column/row".
Example: tclsh tile-logs-to-sqlite.tcl /home/user/tile_logs /tmp/out.sqlite
"10/547/365,10/511/340,10/536/349"
```

In the following Figure 7.2.2.1, structure of spatialite database containing log access data is presented.

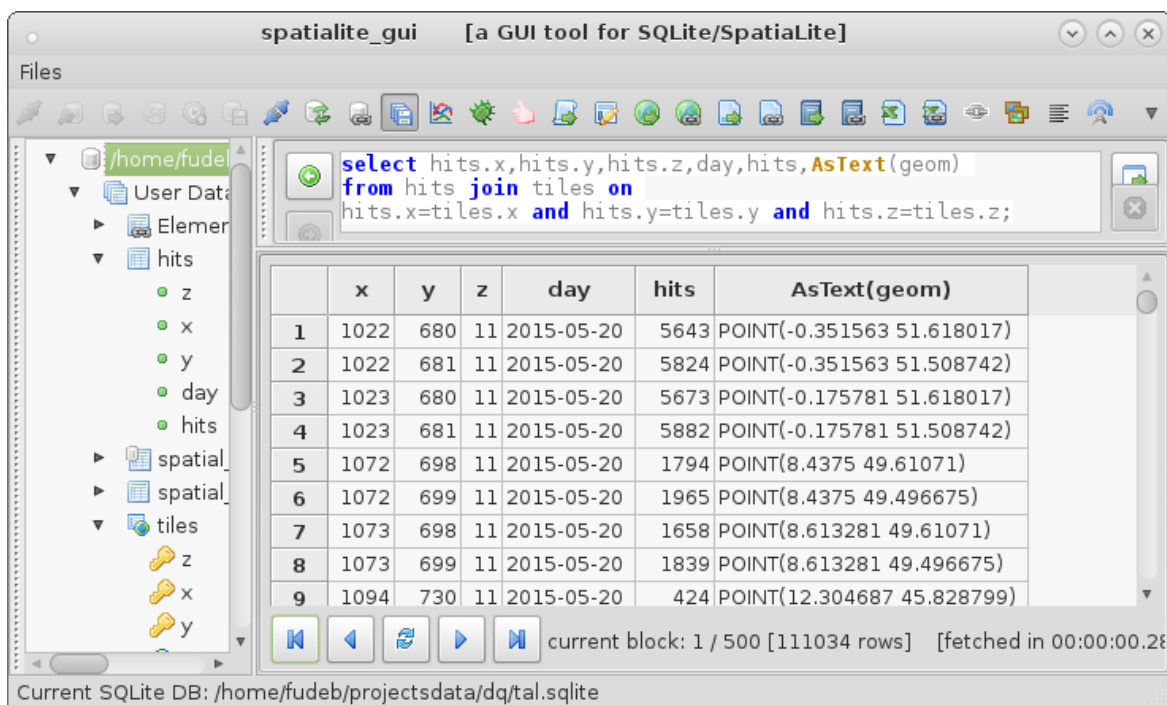
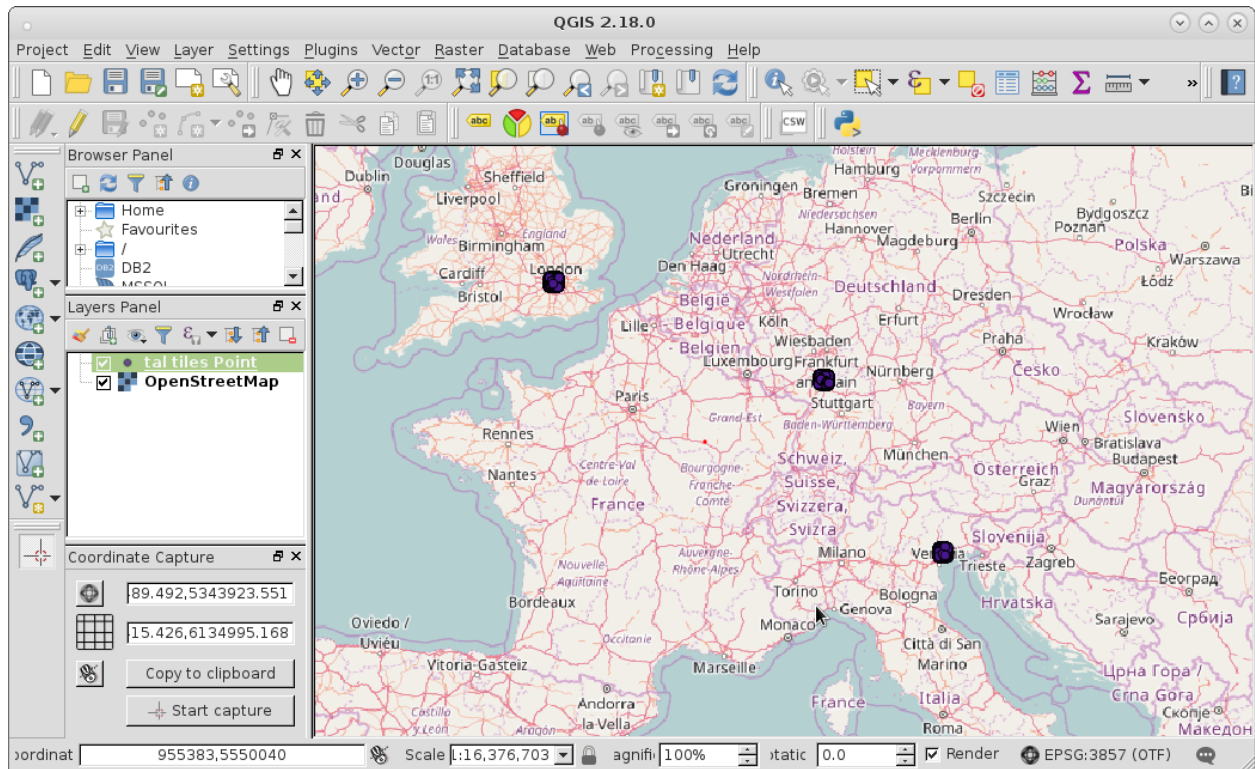


Figure 7.2.2.1. Structure of spatialite database containing log access data.

Currently, it covers London, San Dona, and Heidelberg areas. Torin will be added soon. Database (file tal.sqlite.zip 3.6 MB) could be downloaded from SDAT's data directory (<http://sdatt.n-kov.com/data/>).

Spatialite database is useful for research: it is suitable for querying, everything is packed into single file (thus, it could be easily sent by email or copied into flash drive), zipped database could be processed without unpacking. Spatialite does not require an installation/deploying and could be started from small executable file. Spatialite database could be added as layer by a GIS application. In the following figure, tal.sqlite database is added to QGIS.



*Figure 7.2.2.2. Access log database displaying in QGIS. OSM is used as a background map*

OSM full history dump is another data source. Most of tools for processing OSM data are not suitable for history dump. In order to convert data from dump to spatialite database readOSM and spatialite\_tools were modified(see SDAT's src directory). The output database is as follows.

spatialite\_gui [a GUI tool for SQLite/Spatialite]

Files

select \* from osm\_nodes where rowid<100

	node_id	version	timestamp	uid	user	changeset	filtered	Geometry
1	163292	1	2007-09-29T22:23:04Z	573	jörg Ostertag	535125	0	BLOB sz=60 GEOMETRY
2	163292	2	2007-10-10T00:51:52Z	11008	spacer1971	91672	0	BLOB sz=60 GEOMETRY
3	163292	3	2010-06-04T17:29:10Z	136321	Teddy73	4902177	0	BLOB sz=60 GEOMETRY
4	163293	1	2005-08-09T18:16:32Z	229	LA2	233	0	BLOB sz=60 GEOMETRY
5	163293	2	2007-10-10T00:51:52Z	11008	spacer1971	91672	0	BLOB sz=60 GEOMETRY
6	163293	3	2008-07-30T20:02:43Z	42123	Ropino	630605	0	BLOB sz=60 GEOMETRY
7	163293	4	2010-06-04T17:29:07Z	136321	Teddy73	4902177	0	BLOB sz=60 GEOMETRY
8	163293	5	2013-10-11T21:30:25Z	165061	mapper999	18300094	0	BLOB sz=60 GEOMETRY
9	163294	1	2005-08-09T18:16:54Z	229	LA2	233	0	BLOB sz=60 GEOMETRY
10	163294	3	2010-06-04T17:21:23Z	136321	Teddy73	4902177	0	BLOB sz=60 GEOMETRY
11	163294	4	2013-06-09T06:33:00Z	212111	okilimu	16477213	0	BLOB sz=60 GEOMETRY
12	163294	5	2014-04-29T13:53:23Z	96069	theophrastos	22023646	0	BLOB sz=60 GEOMETRY
13	163295	1	2005-08-09T18:16:56Z	229	LA2	233	0	BLOB sz=60 GEOMETRY
14	163295	2	2010-06-04T17:21:25Z	136321	Teddy73	4902177	0	BLOB sz=60 GEOMETRY
15	163295	3	2014-04-29T13:53:23Z	96069	theophrastos	22023646	0	BLOB sz=60 GEOMETRY
16	163296	1	2005-08-09T18:16:58Z	229	LA2	233	0	BLOB sz=60 GEOMETRY
17	163296	2	2010-06-04T17:20:59Z	136321	Teddy73	4902177	0	BLOB sz=60 GEOMETRY
18	163298	1	2005-08-09T18:17:03Z	229	LA2	233	0	BLOB sz=60 GEOMETRY

current block: 1 / 99 [99 rows] [fetched in 00:00:00.008]

Current SQLite DB: /home/fudeb/projectsdata/dq/ManheimHeidelberg.sqlite

Figure 7.2.2.3. Structure of OSM historical spatialite database (spatialite gui)

The database contains multiple versions of main elements (nodes, ways, and relations). It reflects OSM model in SQL representation. In QGIS, different versions of a node could be presented as follows.

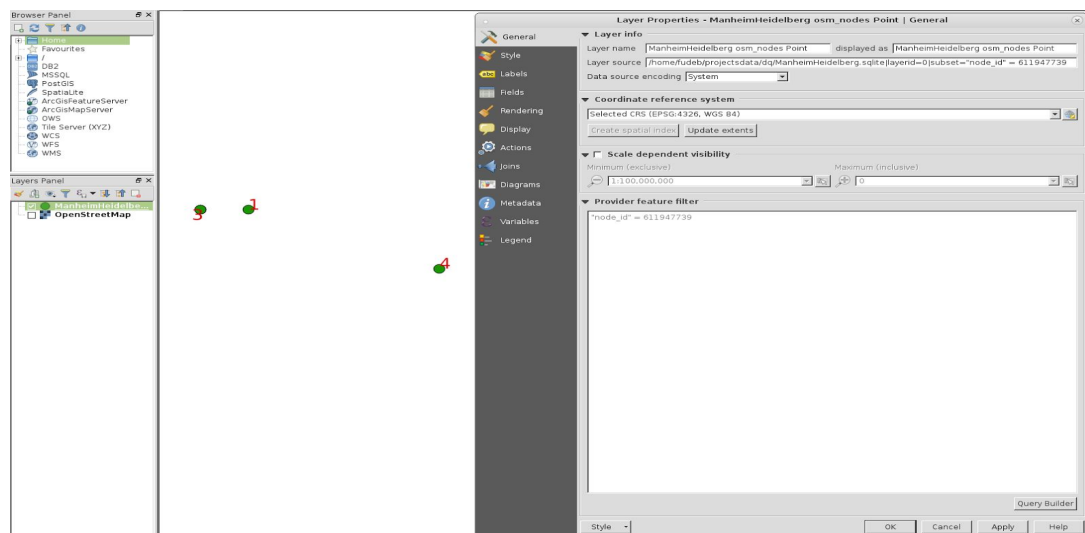


Figure 7.2.2.4. Different versions of an OSM node (QGIS)

It was disclosed recently that tags and way/relation with nodes data are not represented correctly. We have decided to replace converters and rewrite them from scratch, since further modification of source code written in C looks unproductive.

The second block of SDAT, contains scripts and queries for calculating multiple parameters allowing the user assess spatial data. The following example calculates such parameter:

```
SELECT printf('%.1f',avg(maxver)) FROM  
(SELECT max(version) as maxver FROM osm_nodes WHERE  
MbrWithin(Geometry,BuildMbr($x1,$y1,$x2,$y2)) GROUP BY node_id);
```

This script calculates an average maximal version of nodes inside a bounding box. This parameter is used as indicator of quality of OSM data. Currently 62 parameters are defined. The list will be extended.



## **Appendix 7.3**

### **Details about the OnToMap Ontology**

## Appendix 7.3 - Details about the OnToMap Ontology

### Data integration roles of the OnToMap Ontology in the WeGovNow platform.

The OnToMap Ontology has been designed having in mind two data integration models within the WeGovNow platform:

- The first one presupposes that a direct mapping between the concepts of OnToMap ontology and the categories/schema/tables/... of the WeGovNow applications is defined. This type of integration is suitable for those applications that adopt a stable conceptual model of their own domain (e.g., FirstLife could be a good candidate for this approach).
- The second one is loosely-coupled and it is suitable for those WeGovNow applications that do not stick to the details of the OnToMap ontology (e.g., GeoKey could be a good candidate for this option). In order to manage a dynamic conceptualization, which is based on the assumption that new data categories might be freely defined by users, without taking formal ontology requirements into account (e.g., semantic consistency), the OnToMap ontology includes concept “AtomicThing”, which represents any data item tagged by category without requiring the explicit introduction of the category in the OnToMap ontology.
  - E.g., if the users of a WeGovNow application define a set of categories, such as MyServices, Facilities, etc., all the instances of such categories (i.e., all the data items tagged using such categories) will be handled in OnToMap as instances of concept “AtomicThing”, whose category properties are MyServices, Facilities, etc..
  - AtomicThing inherits basic properties from “SchemaThing” (e.g., the external\_URL of the data item, the provenance, etc.) and has a data property “hasCategory” that specifies the name of a category to which the item belongs. In conformance with some representations, e.g., the one adopted in OpenStreetMap, an individual might have more than one category tag; this is represented by allowing that an AtomicThing instance has more than one hasCategory property.
  - The ontology does not explicitly associate AtomicThing to Geometry because the instances of the concept could be either geographical or non-geographical data items. However, as explained in Section 2.6, crowdsourced data are acquired by the OnToMap Logger as GeoJSON Features. Therefore, the geometries associated to such objects can be verified by the Logger itself and do not need to be modelled in ontology.

Figure 7.3.1 of this deliverable shows a portion of the OnToMap Ontology concerning “SchemaThing” and “AtomicThing”. Moreover, Table 7.3.1.1 below describes the properties of the two concepts in detail.

*Table 7.3.1.1 Definition of the properties of concept “SchemaThing” (and “AtomicThing”).*

<b>SchemaThing properties</b>	<b>Linguistic descriptions of the properties</b>
hasName	The name of the instance
hasDescription	The description of the instance
hasURL	A URL related to the instance; e.g., to store the link to the web site of a place
hasExternalURL	The URL pointing to the complete representation of the instance in the source application
<i>hasCategoryName</i>	The name of one of the categories the instance is associated with ( <i>only valid if the instance is an AtomicThing</i> )
isValidFrom	The moment when the instance becomes valid
isValidTo	The moment after when the instance is not valid
hasOriginalProvenance	The provenance of the instance at the moment of its creation
hasCurrentProvenance	The provenance of the instance defined after its last modification
hasRevision	A revision of the instance
hasCertificationAuthority	The certification authority that confirmed that the information related to the instance is correct
hasCertificationDate	The date of the certification of the instance

#### **Status of development of the OnToMap Ontology.**

The OnToMap ontology is under development and it is going to be updated in order to take the information requirements of the WeGovNow pilots and WeGovNow applications into account. The current version has been developed by taking the following types of information into account:

1. Available Open Data from Città di Torino and from Città di San Donà di Piave. The integration of Open Data from the City of London is ongoing work.
2. Portion of the Schema.org domain conceptualization currently adopted in FirstLife. We decided to adhere to this representation in a fairly strict way because Schema.org is a standard data representation adopted by main search engines, such as Google and Bing. It comes from an agreement reached after having analyzed a large number of alternative representations and, being pushed by widely used search engines, we believe that it promises to become a “de facto” standard for the internet.
3. INSPIRE European directives for the publication of Open Data: we checked the correspondance between the Schema.org data representation to the INSPIRE specifications in several aspects, in order to take into account such directives for the publication of European Open Data. Considering the fact that the largest portion of INSPIRE directives concerns geographical Open Data (such as landscapes, etc.), we focused on the portion of the INSPIRE conceptualization of interest for WeGovNow:
  - a. Some representation features are equivalent (e.g., the representation of dates and times) and can thus be directly used in the ontology.
  - b. Following state of the art in linked data representation and exploration, we have replaced other INSPIRE specifications in order to take into account emerging standards for data interoperability. For instance, we have discarded the INSPIRE representation of geometries in favor of the Geometry representation used in the GeoSPARQL (<http://www.opengeospatial.org/standards/geosparql>) language for querying RDF data repositories , as this is the language emerging as a standard for querying linked data and is supported by the current engines for linked data manipulation (triple stores).
  - c. We discarded the INSPIRE “event” concept, which refers to atmospheric events, explosions, and other types of (intentional or unintentional) disastrous events, that are out of the scope of WeGovNow. The “event” concept in the OnToMap ontology describes public and private events organized by people for entertainment, sport, learning activities, and so forth.
  - d. We considered the INSPIRE directives for the management of the provenance of data through the specification of the “Lineage” metadata (<http://inspire.ec.europa.eu/glossary/MetadataElement-Lineage>), which describes the “what, when, how, etc.” of a data source in a textual format. However, we refined the specification of provenance in order to deal with the dynamic nature of crowdsourced information. See below.
4. W3C PROV model for modeling data provenance (<https://www.w3.org/TR/prov-dm/>). We checked the specifications defined on the basis of the previous sources against the W3C model to see whether major differences existed in the representation of provenance information. Basically, the W3C model introduces the concept of Activity,

aimed at describing which type of activity is performed to derive one datum from another one (e.g., translation), and so forth. For the requirements of the WeGovNow platform this type of information is too detailed and thus we discarded it.

The OnToMap ontology supports a machine-readable specification of provenance under the possibility of revising individual data items belonging both to crowdsourced data as well as to Open Data. In general, it might be possible to specify provenance at different granularity levels; e.g.:

- provenance of a whole data set. This is suitable for the publication of archives of Open Data authored by the same entity. It is covered by the INSPIRE lineage meta-data.
- provenance of an individual data item: in case of crowdsourcing, a specific author might upload a data item and, possibly, later on other authors (or him/herself) might update it. This generates a streamline of revisions, each one authored by a different person in a different date.

In order to model both cases we have introduced the representation of provenance depicted in Figure 7.3.1 of this deliverable. The provenance is modeled as a concept in order to support the description of instances of provenance that can be associated to multiple data items, without repeating the descriptions for any item. In the OnToMap Ontology, Provenance is associated to SchemaThing, in order to provide any information item with a provenance specification.

Specifically:

- SchemaThing is associated to an “original provenance”, through the “hasOriginalProvenance” relation, to specify the provenance of an item at creation time.
- Moreover, SchemaThing is associated to the “current provenance” in order to describe the current authoring data.
- Notice also that SchemaThing is associated to Revision through “hasRevision”, which makes it possible to reconstruct the history of revisions of information items, each one associated to its own provenance (“hasOldProvenance”).

Table 7.3.1.2 below describes the Provenance concept in detail. Provenance has several properties in order to support the tracking of author, version of the data item, origin application/data source (for crowdsourced information/Open Data), licence of usage, etc.; moreover, it has the Lineage property to support the storage of metadata about Data Sources.

*Table 7.3.1.2 Definition of the properties of concept “Provenance”.*

Properties of Provenance concept	Linguistic description
hasAuthor	The author of the instance
hasPublicationDate	The publication date of the instance

hasLicence	The licence of use of the instance
hasCredits	The credits of the instance
hasLineage	A textual description of the source of the instance (INSPIRE Lineage metadata)
hasOpenDataSource	A reference of the source Open Data set, if the instance belongs to an Open Data set
hasVersion	The version of the instance (e.g., in case of multiple versions of Open Data)
hasSourceApplication	The source application, if the instance was crowdsourced

## **Appendix 7.4**

### Temporal indexing of urban entities

# TEMPORAL INDEXING OF URBAN ENTITIES: BUILDING A COLLECTIVE CALENDAR OF CITY ACTIVITIES

2017-01-30

Department of Computer Science  
University of Turin, Italy

*In this document, we present a general framework to address the temporal dimensions of urban entities. It highlights the features and issues related to the interpretation of times and timings of entities. The analysis is followed by the definition, implementation and experimentation of a temporal indexing system, in the perspective of the building a web-based global calendar for city activities.*

## 1 INTRODUCTION

Nothing is forever but still most of the web applications do not address dynamics. In the context of applications for cities, such as online urban communities, dynamics cannot be overlooked but should be supported as being the foundation of coordination, collaboration and planning among local actors. Urban entities are in fact tightly bounded to city dynamics: to social and cultural phenomenon and to city services and activities coexisting in the same space. In this regard, there are specific issues related times (the life cycle of entities) and timings (the operational times of entities) in the city.

Specifically, times are addressed by almost every commercial technology enabling application to support the stratification of information during time. In particular, most of the major applications for supporting activities in the city enables time queries, time views of data but timings (recurrences and times of activities) are still not supported at the city scale.

For instance, OpenStreetMap (OSM) is the most popular and effective web volunteering geographical information system (VGI), collecting contributions around the globe. Nevertheless, the representation of geographical entities is static and this represents a problem in civic social networks based on interactive maps where entities are accessible through a time window.

One dimension of time is the distinction between a place's times (when it starts and end to be) and the timing of the activities it hosts. For instance, there are stable entities providing their functions with a specific timing, such as openings and vacations, entities do not stop to be but the timing where they operate is much more relevant in the way we interact with them (is a closed store a resource?). The timing of activities has a strong impact of the conception of what and when activities can be done and therefore it is not a dimension which can be overlooked and flattened to entity existence. Moreover, there ephemeral entities which are even physically present with a certain timing in different places such as market stalls moving around the city or recurrent events such as music or religious festivals.

In this perspective, the collection of city entities is a unique collective global “calendar” defined by all local actors and, at the same time, the foundation of their activities. In this regard, in order to enable technology to represent this global calendar with the times and timings of city entities we need to address issues related to representation and querying of timings (performance wise), and to user interaction with such huge set of data. In particular, the proposed framework addresses:

1. Which are the relevant properties of temporal representation?
2. Which is the nature of the entities
3. How to extend common databases to represent and query timings directly at database level (without relying on software business logics) in a scenario of a global set of entries
4. How to support querying a global calendar of all city entities and activities by heterogeneous users

The approach proposed in this document starts from an analysis of urban entities as social artefacts. Urban entities are the results of a network of activities at local level, involving heterogeneous groups of actors. The time dimensions of urban entities are therefore interrelated with actions and services, processes at urban level as part of the common playground among local actors. From this perspective, urban entities are commons (Ostrom, 2015) shared by institutions, citizens and any other local actor, and in general involved in the orchestration of urban activities, urban planning and part of the mutual understanding of the urban space.

Thus, the entanglement of urban entities in any other aspect of city life makes the definition of a commons representation of the dynamic feature of urban entities itself a challenging issue. In fact, from a theoretical perspective, the multi-dimensionality of temporal features of entities it is not a new issue in the database research community. The best example of consolidated work is TSQL2 (Snodgrass, 2012), it is a consensus extension of SQL for temporal structures meant to address time in a very general way. On the other hand, real applications require enterprise solutions supported by most used database. Just to provide an analogy, in the field of geographical information systems libraries such as PostGIS successfully enabled developer to work with geographical entities using a widely used database such as Postgres. In particular, in the case of PostGIS what really matters were not accuracy or performances of the library but rather its accessibility to developers. On the contrary, even if TSQL2 can be considered a mature approach (1993) based on a strong theoretical framework with the backup of the major experts of the field it still just a theoretical work.

Following this lesson, this contribution does not provide a theoretical framework but a light-weighted ready-to-use fast solution which can be implemented in the most used databases and ORMs based on standard languages.

The rest of the contribution analyses the scenario highlighting challenges and existing approaches. Section 3 presents the hybridization of CRON language for job schedules used in operative systems and ICAL standard for events, its translation in database properties and query language. Section 4 shows applications of such language and discuss its limitations. Section 5 presents the experimentation results and the future extensions. Conclusions ends the document.

## 2 SCENARIO

Representing urban entities is commonly reduced to geographical features. This simplification leads to a static representation, a snapshot, nevertheless it is a suitable solution in most of the common applicative cases. For instance, the many geographical tools for web applications such as LeafletJS and OpenLayer used to collect geographical features do not support temporal information of any kind. Furthermore, technologies

are being updated to dynamic layers of data streams such as environmental sensors, still avoiding to address the issue of temporal features of urban entities.

Supporting collaboration, cooperation and planning requires the idea of an evolving world in which the dynamics in term of timing and coordination play a very important role in the success of initiatives. For instance, the scheduling of the occupation of a public square prevents having a strike, a concert and a sport event at the same place at the same time.

Urban entities as any social artefacts (Searle, 1995) (Ferraris, 2013) have a lifecycle. In this regard, along with the classical challenges of dynamic systems there are specific issue related to observability of such entities. In particular, non-formal entities (not formally defined by local authorities with constitutive actions) being dependent to the establishment of practices have a very fuzzy beginnings and ends. For instance, it is possible can count several important phases of a urban entity such a business centre such as the project, the first contract with a company, the beginning of the construction, the opening, but none of all can be considered the beginning of the place itself.

Furthermore, urban entities are commonly providers of services at local level. The temporal availability of service availability adds an extra temporal dimension in addressing dynamic entities, in particular in case of temporal queries. For instance, the relevance of a closed pharmacy can vary according with the use of the information, is a user looking to the commodities in our neighbourhood or we are in time of need? Moreover, past entities are not just not existing anymore but leaving a trace as impact on the perception of what can or cannot be. On the other side, planned entities which are known to be are “potential” presence on the territory affecting greatly the overall coordination. For instance, public infrastructures have a huge impact on the city landscape even before being realised such as an underground line which result on changing the expansion directives of cities, real estate market, and public/private investments years before the actual opening.

Summarising, addressing the dynamic of urban entities leads to specific issues related to the very nature of social artefacts and urban dynamics:

1. Fuzzy timings, the impossibility to establish a change of phase in an entity lifecycle
2. Worktime of urban services, urban entities are not always available or operating but they still exist
3. Traces of past activities and entities, what was matters
4. Planning the future, potential entities have their own impact even bigger than what they will actually have

Relating to the dynamics of urban entities, there are many other issues related to keep track of changes of shape and location, change of functions of urban entities, etc. All of them require the support of temporal features.

### 3 MODELLING URBAN ENTITIES

About the fuzzy timing, the establishment of entity phases are mostly due by institutional actions or by convention. In general, there is not a way to catch the exact beginning and the end of social artefact because the very nature of social artefacts. In fact, we can address this issue only by relying on evidences provided by users. In this regards, technology can support the emergency of social artefacts widening the collective awareness about such entities. Still, urban entities as social artefacts should be managed considering that there is not a “right” or “rightful” representation because social artefacts are just not fully objective (Ferraris, 2013).

Moreover, the city is made of an infinity of social entities which are not always shared by the most of the citizens or significant for their activities. For instance, community places are of this kind, they have special meanings and uses which are established through community practice only, such as a spot in a public park used to host the neighbourhood party. We address as urban entities only what is significant and relevant for most of the citizens. This very simple description does not lead to a computable mechanism to “catch” urban entities from a sea of information such as social media, but in our opinion makes this issue falling in the field of crowd sourcing.

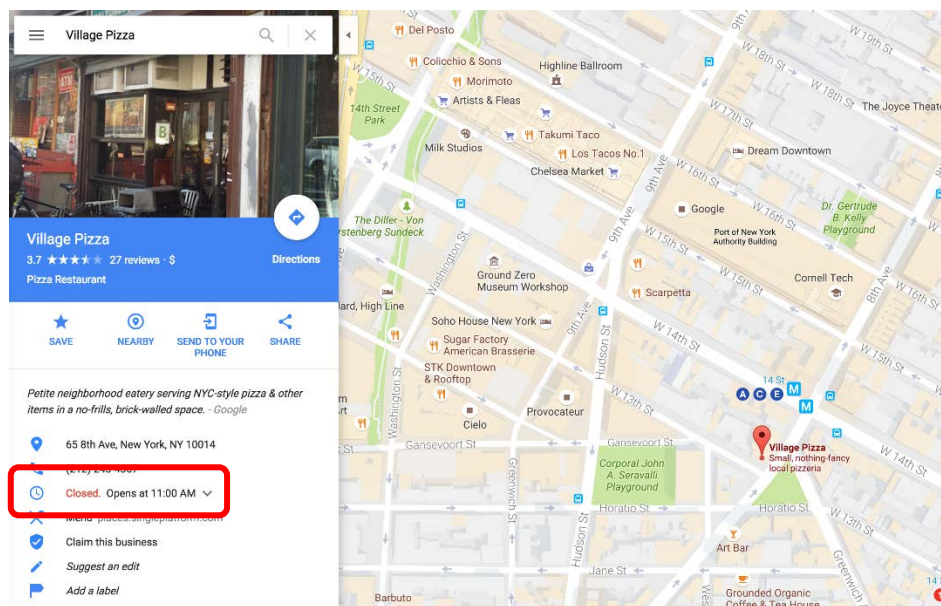
Given this epistemic assumption, we consider citizen as our platform users the sensors for collecting urban entities. In order to collect urban entities from users’ contribution we establish a set of entities at the granularity we wish to catch, for instance events, groups, places.

### 3.1 INTERMITTENT ENTITIES

Services provided by urban entities may not be always available. This may not be true in general but urban entities are commons and the interactions with urban entities falls in the problem of commoning (the use of commons in coordination of local actors) (Bollier, 2014). Therefore, urban entities seen as common resources to be used in an orchestration require to take into consideration the specific limitations of resource availability.

In more simple words, the working time of services in the city has a major impact on our daily organization of activities. Access to the dynamic of urban entities can be approached: 1) at visualization level (opening time in google maps); 2) considering the timing of the availability of services as a relevance metric to filter urban entity.

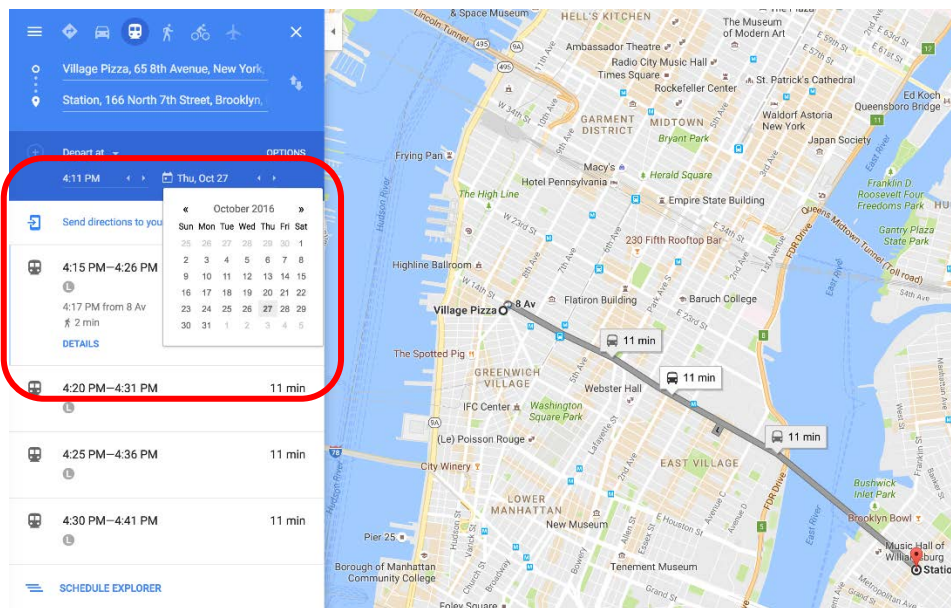
The first approach considers the service availability as a property of the entity (e.g. opening time). This approach is implemented with is a dynamic visualization of the timing, for instance, the opening time of a store can be presented as a simple “open” or “close” according with the current date. In particular, it solves the problem of representing service timing in a static setup (snapshot) thanks to an assumption: the current time is what users are interested in, for instance see Figure 1.



**FIGURE 1. GOOGLE MAPS KEEPS THE OPENING TIME BUT THE INFORMATION IS PRESENTED AS AN EVALUATION, CONSIDERING THE USER’S CURRENT TIME.**

The second approach is based on the necessity to keep the quantity of presented information low, therefore if a service is not available it should not be a “suggested” to users. In particular, this approach works along with ranking algorithm based on distance, preferences, popularity introducing one objective parameters in ranking (and filtering) results. If results are excluded by time queries, we can talk about a dynamic representation supporting time-based views.

The two approaches encode two complementary ideas: a scope and a point of view. Given a specific query, the scope (time window) can be narrow or broad accordingly to the quantity of information keeping the visualisation informative and manageable at the same time. Within a time window, the possible points of view are multiple. The choice of the point of view can be transparent to users, like in Figure 1, or can be bounded to the temporal scope of the view. For instance, routing application provide the possibility to customise a departure time and date, see Figure 2.



**FIGURE 2. ROUTING FEATURE PROVIDE TO USER THE POSSIBILITY TO CUSTOMISE THE DEPARTING OR THE ARRIVING TIME, THIS INFORMATION IS USED FOR INSTANCE TO EVALUATE TRANSPORTATION OPTIONS.**

Summarising, there are at least two aspects which need to be considered in order to evaluate temporal entities: the scope is needed to filter entities according to their valid time; the user’s temporal point of view is required in order to evaluate the timings. All applications supporting temporal features implement a theory of scope and/or temporal point of view as explicit or implicit assumptions.

### 3.2 EPHEMERAL ENTITIES

A new kind of complexity is given by planned or intermittent entities: ephemeral entities. Planned entities are very important in the city organization, let us consider festivals for instance which with a short duration of few days requires months of organization involving many local actors in many locations. Intermittent entities are those entities “happening” with regularity such as markets. The impact of intermittent entities involves the organization but also: the timing when provided services are available and the unavailability of other services which are replaced by the intermittent entities. For instance, street markets and parking slots are mutually exclusive.

The extreme consequence is stressed by the phenomenon of ephemeral cities: giant settlements appearing and disappearing with an interval of years in case of religious or cultural events (Kehoe, 2011). In this cases

entire cities with their system of infrastructures and services are planned, organised, implemented, and finally dismantled for existing only for a short period of time but influencing an area for years.

The representation of ephemeral entities can be done with digital maps. In particular, digital maps can support planning and documentation of ephemeral entities if including a temporal support. Digital maps can work as exploration tools for what was in the past and what will or can be in the future.

What matters specifically in case of ephemeral entities are the connections among local activities and areas as complement of planned urban entities. For instance, the difference between a street market stall and common grocery store is not the timing of provided service but the orchestration of activities before and after required to setup the service such as setup the market stalls and clearing the market area after, in addition to the mutual exclusivity use of the same area with other services.

Ephemeral entities are an extreme case of urban entities: some urban entities are planned, other are incidental but all of them have a live cycle connected to urban activities. Since urban entities are the results of initiatives, and can trigger many other initiatives at different timing, in order to collect effectively urban entities, it is required to extend the efforts to the related activities. In other words, the granularity of entities in an urban calendar should be fine grained at level of activities.

### 3.3 ASSUMPTIONS ON TIMES AND TIMINGS

Time is one but the temporal features can be many: the duration of an event, the time when the event was planned, the time when the event become public. In addressing time, there are issue related to time semantic, how we should evaluate the things we know about an entity if we look at it from a different time or if we consider at a different scale. In the context of urban entities, which are the relevant temporal features? What are the relevant questions about the duration and the repetitions of urban entities?

The concept of timing in real applications, time repetitions, is bounded to the concept of time granularity: time deltas between repetitions. Time granularity is a very complex field: theoretically is the study of relations between class of time intervals. A calendar is a specific setup of interval classes and relations. For instance, we can count many “official” calendars across the world such as the western calendar or the Chinese calendar), school calendars, work calendars with differences in term of durations (work week of 5 rather than 7 days), general alignment (beginning of the year), or specific granularity levels (“semesters” of 4 months in school calendars).

Time granularity is being already addressed (Snodgrass, 2012) making possible to define new calendars and mapping between calendars. In a context of urban activities everyone refers to the national plus some common notions of festivities and work time (weekly holidays). In this context, a general framework is not strictly required instead a pragmatic approach can be implemented with enterprise technology rather requiring the implementation of theoretical a language such as TSLQ2.

A pragmatic approach does not save us from the problem of time semantics, in particular when comes to representing information at different granularity. In fact, timed information semantically dependent to the granularity of their time description. For instance, a salary amount can be per month, week or year. In any case, a change of time granularity (in information query) may or may not affect the outcome value.

This phenomenon is called *telicity* and it is mostly applied in studying the semantics of verbs and verbal phrases. In temporal database, *telicity* influences the transformation record values according with the change of time granularity (Khatri, Snodgrass, & Terenziani, 2009). The semantics of time queries is dependent to *telicity* of entities. In our context, the “value” that we are referring to is the validity of a result is a time query.

For instance, an opening time of from 8:00 to 16:00 from Monday to Friday defines the availability of a service in term of day of the week and hours. The timing is given by the combination of two expressions defined at different granularity: day and hours. Considering a query about the opening time on Monday and on Sunday, in the latter case we can safely say that it our shop is closed but in case of Monday we cannot be certain since during the day there will be both open and close hours. In fact, the granularity of the combination of the two timing expressions is hours: a shop is opened for 8 hours each 16 hours with two exemptions each 5 openings. On the other hand, in case of a query for a specific minute such as Monday at 8:59 we don't have any problem scaling down the granularity.

This example shows how tricky timing in term of query resolution, even trivial case may rise unexpected complexity. Moreover, it highlights that the telicity has an impact of properties of timed entities. In particular in case of the previous example, we see different semantics according with store being liquid (Bettini, Wang, & Jajodia, 1998) or not in respect of its opening time. Liquidity is the property of preserving value changing the scale upward and or downward. In the previous example was easy to see as opening time have downward but not upward liquidity.

This issue cannot be solved with a data schema, since schemas do not represent semantic of information cannot be solved with a priori logic if we do not fix the domain of application and certainly we cannot ask users to define semantic of time and timing features (Bettini, Wang, & Jajodia, 1998). Furthermore, in case of punctual time information (timestamps) we may have other properties such as persistence: an entity lasts unless something changes.

In our context, the conservative position is assuming telicity for each entity and timing expression, in other words we are not sure when we may assume upward or downward liquidity or persistence: scaling entities upward or downward or persisting entities in case of punctual time information (now is open) in the future may mislead users introducing false positives.

Summarising, a pragmatic stand oriented to real-life generalist applications should consider the issue of telicity or the lack of information carefully, designing a coherent set of assumptions about persistence, upward and downward liquidity and other possible point-based or interval-based semantics.

In this regards, our approach is to exploit users' input (on a timeline) about the relevant time window (and granularity). In fact, since the biggest issue is filtering information, we assume avoid false positive presenting only the entities whose timing descriptions are defined in the input granularity. This may rise in hiding important information but since the timeline is interactive, users can still access to information by switching the time granularity.

### 3.4 TIME FEATURES AND TIME-BASED TECHNOLOGIES

Entities may have multiple time dimensions. For instance, the classical dimensions in databases are between transaction time and valid time: the first is the time when an information becomes available in a database, the latter is the time in which the information is true. Transaction time is mostly punctual and it assume the persistence of knowledge: knowledge is preserved until it changes. Valid time are commonly expressed by a couple of dates (start and end dates). The semantic of valid time is usually implemented by business logic field by field. In commercial databases up to today there is no generic support for time repetitions. In few rare cases it is possible to express a time schedule: repetition of punctual times and not of time intervals.

Even so, there are few very handy web applications supporting times and timings such as web calendars or events platforms. In those cases, the solutions are protected by their companies. What we can know is that the few open source solutions rely on business logics and data visualization in a context of very limited

quantity of information (a user's events on a calendar are not much). In particular, the visualisation of web calendar comes in help highlighting a specific portion of time. This enables many possible heuristics such as dereferencing periodicity in a chain of single events in a reasonable wide time window.

Considering the goal of building a global calendar of millions of events in a territory such as a city, we seek a production-ready solutions meant to support numbers in real case scenarios. The most used formalism to express recurrences in software and database are the CRON expressions, regular expressions meant to be fasted computed with low resources by a job scheduler.

The complexity of cases in real life requires an expressivity much deeper than what may be requested by technical software such as job schedulers. In this sense, ICalendar standard represent the consensus about a common language for calendar, defining entities, recurrence expressions, exception, alerts, time zones, durations and other specific features of real life applications.

### 3.5 CRON EXPRESSIONS

CRON is a family of time-based job schedulers used in Unix operating systems. CRON jobs are a set of CRON expressions representing when a task should be triggered. The timing is expressed by a regular expression (Wikipedia, n.d.) composed of 5 or more substrings defining a set of time points at each time granularity.

```
# _____ min (0 - 59)
# | _____ hour (0 - 23)
# | | _____ day of month (1 - 31)
# | | | _____ month (1 - 12)
# | | | | _____ day of week (0 - 6) (0 to 6 are
# | | | | | Sunday to Saturday, or use names)
# | | | | |
# | | | | |
# | | | | |
# * * * * * command to execute
```

For instance, a command that should be executed each day at 3 a.m., such as a database dump can be expressed as follows:

```
0 3 * * * # every day at 3:00
```

A job that should be executed once a week (every Saturday at 3:00) such check-up of the file system can be expressed as:

```
0 3 * * 6 # every Saturday at 3:00
```

A CRON expressions can represent multiple recurrences, for instance at 3 a.m. and 1 p.m. during workdays on odd months:

```
0 3,13 * 1/2 1-5      # from Monday to Friday at 3:00
                        # and at 13:00 of Jan, Mar, May,
                        # Jul, Sep, Nov
```

In particular, the syntax allow special symbols (\*, / -) representing respectively: no restriction or always (\*), concatenation (3,13 at 3h and 13h), increment of a number  $n$  (e.g. /2 every 2 occurrences) and interval (i.e 1-5 from Monday to Friday).

Notice that the third and last field are both at the same granularity since week days are days of the month too, therefore they can be used together but in case of overlapping the result is not to be duplicated but the union of the two sets. For instance, considering \* \* 1/2 \* 1-5 says each 2 days starting from the first day of the month and every day from Monday to Friday, the result will be every day from Monday to Friday plus all odd Saturdays and Sundays. If one of the two is defined, then the other is ignored.

There exist many versions of CRON expressions extending the semantic. For instance, we considered Quartz CRON (Quartz) (Software AG, n.d.) which is a Java implementation of CRON. Quartz introduces two extra fields: seconds and years and more symbols (L # W ?). In particular a Quartz expression can represent schedules like:

```
0 15 10 ? * 6#3      # at 10:15 on the third Friday of
                        # every month
0 15 10 L * ?        # at 10:15 of the last day of
                        # every month
0 15 10 * * ? 2016   # at 10:15 of everyday of 2016
```

Quartz expressivity does not come free, in particular:

- $L$ ,  $\#$  and  $W$  symbols standing cannot be resolved without providing a specific month and year
- $?$  symbol is used to disable “day of the week” or “day of the month” alternatively. Otherwise in Quartz the use of both at the same time may lead to unwanted results.

Quartz notations can actually catch many scenarios we may see in real life. But in order to evaluate a Quartz expression it is required a specific time (a month or a year) because we can’t say which what is the last day or the third Thursday of a month in general, but we need to point a month and year (e.g. May 2016). A Quartz expression cannot be evaluated by its definition but it requires to be resolved with its extensive representation.

In general, CRON expressions are limited to point-based time information, in other words it is not possible to represent durations in any of CRON variants. For instance, it is possible to represent the opening time but not the duration (e.g. from 8:00 until 16:00, 8 hours). Moreover, CRON expressions do not support exceptions such as not during Christmas holidays (from 25<sup>th</sup> of Dec to 6<sup>th</sup> of Jan).

In order to represent durations two CRON expressions are required, one for the starting time and one for the ending time. This semantic cannot be represented straightforward by a database schema and it requires a storage procedure to define how to combine the two records.

### 3.6 ICALENDAR STANDARD

ICalendar is a file format and standard notation for representing events (Oracle Corporation, n.d.). ICalendar is adopted as intercommunication language among mostly enterprise products and web services such as Google Calendar, iCal, IBM Lotus Notes, Microsoft Outlook, Mozilla Thunderbird, etc.

ICalendar define several entities definitions: VCALENDAR (calendar), and entities VEVENT (events), VTODO (tasks), and VJOURNAL (journal entries). ICalendar is relevant because it takes into account a decade of experience of the web (1998 updated in the 2009 and still improved) and industrial requirements of the major software developer.

Concerning the issue of time and timing we can specifically focus on RRULE, the attribute of ICalendar describing repetitions. RRULE value (RECUR) can express repetitions of time intervals and exceptions too. Still, RECUR expressions are much more expressive but there are CRON expressions which cannot be encoded as RECUR, for instance:

```
0 0/2 * 1/3 *          # at 00:00 every two hours
                        # every day of each three months
                        # starting in Jan
```

While CRON expressions are designed to be implemented in job scheduler, ICalendar does not look so, in particular there are not standard libraries and the specifics are only partially implemented by the supporting software, making ICalendar mostly an interexchange format rather than an operational language.

## 4 A DATABASE-ORIENTED SOLUTION FOR TIMES AND TIMINGS

A real world scenario of millions of events suggests the necessity of a solution comparable to common information storage and retrieval in database. Specifically, we seek to represent time and timing as records and be able to query such records in real time, fast and without extra burden for the DBMS and without relying on post processing results crossing field values: we wish to store time and timing as records and be able to query using the same expressivity using relational selection only.

In order to archive this result, we require a compromise in terms of expressivity of the language for times and timings. On step of our approach is to decompose complex expressions beforehand in order to keep the querying simple. The second is to use the same language in both representing and querying time and timing expressions, in order to implement queries with a simple selection operator.

#### 4.1 FROM POINT-BASED CRON EXPRESSIONS TO INTERVAL-BASED CRON MASKS

From a computational perspective, the most convenient starting point is standard CRON expressions, but: can we actually express intervals with CRON masks? As argued previously, CRON expressions are not meant to express intervals, but repetitions of point-base times.

Considering the following CRON expression:

0 8-16 \* \* 1-5

It does not express “from 8:00 to 16:00 from Monday to Friday” but a set of points in time “every hour at 00 minutes from 8 to 16, from Monday to Friday”. In particular, CRON expressions represent quantized time points in 5 granularities, sections of a CRON expression. Following the previous example, we have at each minute 0 of each hour between 8 and 16 included, each day between Monday and Friday included:  $1 \times 8 \times 5 = 40$  moments each week.

The intervals between two moments are given by the granularity of the section. Following, from moments it is possible to infer intervals, for instance 0 minutes is equivalent to the set  $[0,1)$  because it is not possible to express a moment between 0 and 1 minutes. As consequence we can consider quantized intervals rather than just moments. Following, the same example we have 1 minute each hour between 8 to 16 each day from Monday to Friday,  $1 \times 8 \times 5 = 40$  minutes each week.

In this sense, the \* symbol indicates the full interval (60 minutes, 24 hours, one month, one week, one year) according with the position, and the semantics is given by the intersection between the 5 sections.

CRON expressions can converted in 5-tuples of bit masks which are widely supported in commercial databases. The conversion of CRON expressions in bit masks is trivial since standard CRON are regular expressions and each section of a CRON expression indicates a set of a finite domain (minutes, hours, etc.). Following an example:

```
# ┌────────── minutes (1...0 1th bit at 1 of 60 bits)
# │ ┌────────── hour (0010...0 - third bit at 1 of 24 bits)
# │ │ ┌────────── day of month (all 0 of 31 bits)
# │ │ │ ┌────────── month (1010... - odd bits at 1 of 12 bits)
# │ │ │ │ ┌────────── day of week (0...10 6th bit at 1 of 7 bits)
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ │
# 0 3 * 0/2 6          #every Saturday at 3:00 of odd months
# m h d M w
```

The \* symbol is encoded as a set of 0 as special case. Now, we need a way to check the matching between a set of CRON expressions and a query (also a CRON expression). We encoded CRON expressions as a set of 5 bit masks (CRON), using a bit for each. It is possible to query a table of CRON (CRON table) using bitwise operators, given a CRON as query. Given a query CRON  $q = (q\_m, q\_h, q\_d, q\_M, q\_w)$  and a generic CRON  $g = (g\_m, g\_h, g\_d, g\_M, g\_w)$ , the comparison operator  $O$  is defined as follows:

$$q \text{ O } g = \&\&(!q\_i \mid \mid !g\_i \mid \mid q\_i \& g\_i)$$

With  $\&\&$  indicating the conjunction of the disjunction ( $\mid \mid$ ) of the check between  $q$  and  $g$  columns, and  $\&$  the bitwise and. In words, a CRON matches iff the conjunction of the check of each column is true. Each column is matched with the corresponding query section with a disjunction of the following conditions: the CRON has \*, the query has \* or the bitwise and between the two bitmasks is true (the intervals overlaps). Following some examples with generic bitmasks:

```
q = 00101 00000 10101 00101 11111      # query

c1 = 00000 10001 00000 00001 00000
    true  true  true  true  true          # q O c1 = true

c2 = 00000 00000 11111 00000 00000
    true  true  true  true  true          # q O c2 = true

c3 = 00000 00000 01000 00000 00000
    true  true  false true  true          # q O c3 = false

c4 = 11111 01100 01000 11000 00110
    true  true  false false true          # q O c4 = false
```

The column match is true iff one or both masks are 0 or they have at least one bit at 1 in the same position (e.g.  $c4$  do not share any bit at 1 with 1 in the 4<sup>th</sup> column). In words,  $O$  is an extension of Allen's overlap relations  $o$  between intervals to sets of intervals, it is true iff there are overlaps between all sets of intervals at each granularity. see the following example.

```
----      -----      # query q
--  --      -----      # c1 O q = true
      --      --          # c2 O q = false
          -----      # c3 O q = true
```

Why is it possible to query CRON without explicit dates? A CRON expression is an intentional definition of a class of intervals but a CRON mask is an extensional definition of a class of intervals. Encoding CRON expressions in bit masks makes explicit a set description of the class of intervals defined by CRONs enabling querying operations. A CRON is still a class definition, without casting a CRON within an explicit time we can't

calculate for instance all intervals. In particular, we don't have a fixed number of days or Mondays in a month, but until we can express a query with a CRON it is possible to avoid casting CRON.

As previously described, CRON expressions are not meant for representing nothing more than job schedule; we need to extend their CRON expressivity in order to partially cover ICalendar cases.

## 4.2 EXTENDING CRON EXPRESSIVITY

Confronting ICalendar with the software supporting it, it is easy to see that not all possible scenarios are actually implemented and used but there are some basic features which can be found in all calendar applications:

1. Date constraints
2. Granularity
3. Exceptions
4. Occurrences
5. Cardinal recurrences

In order to extend the expressivity of CRON we had to enrich the notation introducing new fields in the CRON table. On the other hand, some notations can be actually addressed simply by encoding the CRON as bit masks.

## 4.3 DATE CONSTRAINTS

CRON as defined do not address times but only timings, there is not a way (in standard CRON) to express a valid time as explicit dates. Moreover, CRON extensions like Quartz support years and seconds, consenting the representation of date constraints but it did not consider year field in CRON. The reason is that regardless of the specific field a CRON needs a boundary in order to be encoded as bit masks: an open set cannot be converted in a fix length bit mask. In order to add a column for years we need to define an upper granularity such as decades, century or millennia. Summarising, we can only push the problem but in the end we still need to handle an open set to fix a constraint.

In order to constrain recurrences is sufficient to introduce a valid from and valid to fields. It is a standard solution in database schema which does not affect performance, in particular it can be a very handy solution to pre filter records and setup a distributed architecture.

Moreover, the introduction of date constraints may solve some issues related to real life cases, for instance *"18:00 of Monday to 8:00 of Friday"* can be easily solved this way. It does not look like but this simple example cannot be represented with a single CRON because hours constraints should not be applied to each day of the interval but respectively 18:00 to Monday (18:00 – 23:59) and 8:00 to Friday (0:00 – 8:00). The related CRON will look like the following:

```
* 18-8 * * 1-5      # from 18:00 of Monday to 8:00 of Friday?
```

This expression makes no sense because of the 18-8 interval, but even without this issue it is still not representable. For instance, we can consider the following revised version

```
* 8-18 * * 1-5      # from 8:00 of Monday to 18:00 of Friday?
```

The semantic of this CRON is not what we expect, in fact the right interpretation is the following: from 8 to 18 (10 hours) each day from Monday to Friday. This example requires 3 standard CRON to be expressed:

```
* 18-0 * * 1      # from 8:00 to 23:59 of Monday
*   * * * 2-4     # from Tuesday to Thursday
* 0-8 * * 5       # from 0:00 to 8:00 of Friday
```

With the introduction of the valid from and valid to we can simply represent the previous example as follows:

```
* * * * 1-5 (2016/10/17@18:00) (2016/10/21@8:00)
```

Still, there is a difference between the two solutions, the first one does not require an explicit date, in the second case we use a date constrain to bind a generic interval description. In the second case the CRON semantics is wrong but thanks to the validity time the result is as expected.

Validity time can be optional (null by default) as it is implemented in some applications like Google Calendar. Since we do not need to cast all possible occurrences it does not endanger performances or rise memory issues.

#### 4.4 GRANULARITY

CRON can represent quantized intervals only, therefore in case of intermediate intervals the only solution is approximation. For instance, if we encode the timing “8:30 – 12:00, 14:00 – 18:30” using the comma operator we have the same issue of previous examples because we can’t represent intervals of half hour:

```
0-30 8-12,14-18 * * *      # 8:00 – 8:30, 9:00 – 9:30...
```

In general, the maximum approximation is the granularity itself, in case of the previous example we should drop the constraints in the minute section and approximate the ending time to next hour:

```
* 8-12,14-19 * * * # 8:00 – 12:00, 14:00 – 19:00
```

In order to use CRON an indexing system the approximations may not be a problem, but in some scenarios we may require extra sensibility. In this regard, there are two possible strategies:

1. Introducing an intermediate granularity such as day sections (“*morning*”, “*afternoon*”, “*evening*”, “*night*”) and the relative column
2. Combining two granularities in one single column, for instance minutes and hours can be replaced both with half-hours (48 bits)

The two strategies can be combined in order to keep timing representations short. Considering the half-hour granularity, the previous example can be represented because the quantized intervals become of 30 minutes.

## 4.5 EXCEPTIONS

A very common case are recurrences with exceptions, in particular with generic exceptions (not just date restrictions), such as holydays. It is interesting to notice that those kinds of exceptions can be expressed as CRON, for instance:

```
* * 25-26 12 *           # from 25 to 26 of December
                           # Christmas holidays
```

The solution we developed introduces the notation of negative CRON, an extra field indicating if the CRON should include or exclude the related event.

```
false * * 25-26 12 *     # Christmas holidays
true  * 8-18 * * 1-6      # Openings 8-18 Mon-Sat
```

In order to use positive and negative CRON expressions it is required to define how to handle the cases where both are included in the results (for instance Monday 25 Dec at 12:00). A simple semantics can include an event if the conjunction of the control flags of the returned CRON is true.

## 4.6 OCCURRENCES

In commercial applications is common to possibly define an occurrence number as limit for repetitions. For instance, each two days for 10 times. This can be handy in case of a fixed set of events scheduled at the same time.

CRON does not support counters but counters are syntactic sugar, explicating the occurrences and indicating an end date is equivalent. Therefore, in this case it is possible to support this feature through a pre-processing phase, which it does not increase the overall complexity of the system.

## 4.7 CARDINAL RECURRENCES

Some recurrences are defined by cardinality, for instance *“each second Sunday of the month”*. Those cases are supported by ICalendar and Quartz but not by standard CRON. In particular, cardinal recurrences are necessary in order to connect days of the week and months, because there is no way to know the exact day of the month in general. In other cases, this notation is syntactic sugar, therefore it can be addressed via pre-processing inputs.

In order to address this feature, it is required a new column for the cardinality of day of the week (5 bits because 5 is maximum of occurrences of a day of the week in a month). Given the previous example in Quartz notation (# indicates the cardinality):

```
* * * * 0#2          # second Sunday of each month
0 0 0 0 1000000 01000    # equivalent bit masks
```

Extending the day of the week with cardinality stresses the use of `*` symbol in queries. In particular we can consider the following query:

```
0 0 0 0 1000000 00000    # q = Sunday

0 0 0 0 1000000 01000    # c1 = second Sunday of the month
t t t t true      true    # q O c1 = true
```

The problem seems to be the lousy semantic of `O` but with the same operator we can express two different queries: “*all Sundays*” and “*some Sundays*”. In order to explicit the “*all*” quantifier the query should be implemented as follows:

```
0 0 0 0 1000000 00000    # q1 = some Sundays
0 0 0 0 1000000 11111    # q1 = all Sundays

0 0 0 0 1000000 01000    # c1 = second Sunday of the month
t t t t true true        # q1 O c1 = true
t t t t true false       # q2 O c1 = false
```

In particular the use of the symbol `*` in a query or in entries causes to skip the check, but the full set mask indicates the full interval as requirement.

## 4.8 LIMITATIONS

There are cases we cannot address with CRON without generating false positive or false negative results. In particular, we cannot address:

- The last operator “*L*” and the negative cardinals (e.g. -1 is equivalent to L, -2 means second to last, etc.)
- The weekday operator “*W*” indicating the closest weekday to a given day of the month

Those operators require to evaluate expressions in a given date. Specifically, we do not know how many Mondays there are in a month (4 or 5?) therefore, we may cannot encode the last or negative cardinals in a bit mask. Moreover, the weekday requires also to know the positions on weeks in a month in order to calculate the proximity.

They may exist particular applicative scenarios strongly requiring those operators but among the examples we collected so far we did not find a case which cannot be converted in the supported notation yet.

## 5 EXPERIMENTS

Since this contribution wishes to address real life applications, in order to stress the expressivity of the system, we collected a dataset of real timings from Porta Palazzo market square in the city of Turin, Italy. Encoding the dataset, we exposed limitations we addressed in the previous section. Following, we stressed the system performance with random generated entries and queries.

### 5.1 CORRECTNESS

We collected so far 233 timing descriptions over a total of a hundred of market stalls. Most of them where very simple but in some cases we found exceptions, seasonal timings, cardinal occurrences and alternate days.

A sampling the most representative cases:

7:00 - 14:00 Mon - Fri

7:00 - 19:30 Sat

Common opening time.

8:00-13:00, 15:30-19:30 Tue - Fri

7:00 - 19:30 Sat

Closing at lunch time.

14:00-19:00 Sat from 1st Aug to 15th Oct

This is an exception to the normal opening during summer time.

8:30 - 13:00, 15:30 - 19:30 Mon-Tue, Thu-Fri

8:30 - 13:00 Wed

8:30 - 19:30 Sat

Half day opening on Wednesday.

9:00 - 13:00, 15:30 - 19:15 Tue - Fri

Closing at 19:15 requires approximation to 19:30.

9:00 - 12:30, 15:30 - 19:15 Mon-Tue, Thu-Sat

Closed on Wednesday and approximation of the closing time.

8:30 - 13:00, 15:00 - 19:00 Mon-Fri, Sat/2

Opening on Saturday only each two weeks.

## 5.2 PERFORMANCES

Tests were run on a document-based database (MongoDB) over more than one million random generated entries. The tests were conducted increasing the number of queries by tenfold recording total, maximum and average time.

1.023.944 ENTRIES			
NUM. OF RUNS	Max time (ms)	Total time (ms)	Average time (ms)
1	2	2	2
10	2	4	0,4
100	2	8	0,08
1.000	3	44	0,044
10.000	3	265	0,0265
100.000	3	2380	0,0238
1.000.000	3	21675	0,021675
10.000.000	5	212000	0,0212

TABLE 1. RANDOM QUERIES EXECUTED OVER A DATABASE OF 1.023.944 ENTRIES, THE AVERAGE TIME IS CLOSE TO ZERO (LESS THAN 0,1 MS).

Following, we repeated tests but firing sets of 10, 100, 1000 and 10000 concurrent queries.

1.023.944 ENTRIES – BATCH OF 10 CONCURRENT QUERIES			
NUM. OF RUNS	Max time (ms)	Total time (ms)	Average time (ms)
1	2	78	7,8
10	2	76	0,76
100	3	333	0,333
1.000	2	1766	0,1766
10.000	2	13799	0,13799
100.000	2	130416	0,130416

1.023.944 ENTRIES – BATCH OF 100 CONCURRENT QUERIES			
NUM. OF RUNS	Max time (ms)	Total time (ms)	Average time (ms)
1	2	3582	35,82
10	2	5919	5,919
100	2	19524	1,9524
1.000	2	127193	1,27193
10.000	2	1203622	1,203622
100.000	6	12437346	1,2437346

1.023.944 ENTRIES – BATCH OF 1.000 CONCURRENT QUERIES			
NUM. OF RUNS	Max time (ms)	Total time (ms)	Average time (ms)
1	2	118111	118,111
10	2	317724	31,7724
100	3	1442373	14,42373
1.000	5	12295172	12,295172

10.000	5	115261069	11,5261069
--------	---	-----------	------------

**1.023.944 ENTRIES – BATCH OF 10.000 CONCURRENT QUERIES**

NUM. OF RUNS	Max time (ms)	Total time (ms)	Average time (ms)
1	5	16643359	1664,3359
10	3	26489348	26,489348
100	4	137238709	137,238709
1.000	5	1350002410	135,000241

**TABLE 2. RESULTS OF THE EXPERIMENTS WITH BATCH OF CONCURRENT QUERIES, WE CAN SAFELY STATE THAT IN THE WORSE SCENARIO WITH 10.000 QUERIES WE HAVE WITH MULTIPLE AN AVERAGE TIME OF LESS THAN 0,2 SECONDS.**

Tests were performed on a VM with 2 cores and 2 gb of RAM only without any database tuning. The overall result is in worst scenario less than 0,2 seconds in case of 10.000 concurrent queries. As far as we know, there are no available results for comparison but in term of web services and considering a common workload of 100 concurrent queries the time is irrelevant (less than 0,2 milliseconds) considering an average time of 70 ms for very fast cached calls such as Google Maps's API.

## 6 CONCLUSIONS AND FUTURE APPLICATIONS

This document addresses the technical issues of representing times and timings of urban entities in order to build a collective calendar of city activities. In particular, the premises of the proposed solution are: an applicative scenario in which there is not a user-based partition of entries such as in calendar applications; an indexing system to compute the relevant entries should not require business logic to run at query time; the solution should be able to represent real cases; the solution performances with a common environment setup should not hindrance the overall performances.

The developed solution is based on the most common formalism for timings, CRON expression, and integrated in order to support time intervals, valid time and the most common features supported in the standard format for calendar for software applications ICalendar.

Along with the discussion of the specific issues of representing timing according with the premises, we introduced a set of solutions, workaround and limitations in the scope of the presented solution. Despite limitations, the solution was proved to be able to represent real cases of timing collected from real commercial activities in a dynamic environment such as market squares.

Following, through a set of experiments with random entries and random queries we tested the solution in a low capacity environment such as a common web server. The experiments where organised as cycles of batch queries fired concurrently. In a common scenario with 100 concurrent access the queries where solved in a fraction of millisecond. Under heavy requests (10.000 concurrent queries) the system archived a maximum an average time of 0,135 seconds without any caching system or database tuning.

In conclusion, the presented solution can be used to represent and retrieve high number of entries with a common database without relying on post processing results. The costs in term of expressivity and approximation compared with the real world examples are shown to be very reasonable. Moreover, it was

discussed a set of possible workaround in order to extend the expressivity or compress the representation according with the application demands.

## 7 REFERENCES

- Bettini, C., Wang, X. S., & Jajodia, S. (1998). Temporal semantic assumptions and their use in databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(2), p. 277-296.
- Bollier, D. (2014). *Think Like a Commoner*. Gabriola Island, Canada: New Society Publishers.
- Ferraris, M. (2013). *Documentality: Why it is necessary to leave traces*. Fordham University Press.
- Kehoe, K. L. (2011). *Burning Man was better next year: "a phenomenology of community identity in the Black Rock counterculture*. California State University, Communication Studies. Sacramento: Diss.
- Khatri, V., Snodgrass, R. T., & Terenziani, P. (2009). Telic distinction in temporal databases. In *Encyclopedia of database systems* (p. 2911-2914). Springer US.
- Oracle Corporation. (s.d.). *ICalendar*. Tratto da ietf.org: <https://tools.ietf.org/html/rfc5545>
- Ostrom, E. (2015). *Governing the commons*. Cambridge university press.
- Searle, J. R. (1995). *The construction of social reality*. Simon and Schuster.
- Snodgrass, R. T. (2012). *The TSQL2 temporal query language* (Springer Science & Business Media ed., Vol. 330). Springer.
- Software AG. (s.d.). Tratto il giorno 2016 da Quartz Job Scheduler: <http://www.quartz-scheduler.org/>
- Wikipedia. (s.d.). *Cron*. Tratto il giorno November 2016 da Wikipedia: <https://en.wikipedia.org/wiki/Cron>