Webservices and cloud communication

Philippe Possemiers, AP Hogeschool Antwerpen, Belgium

Abstract

This course is an introduction to the Ionic framework. Ionic is a powerful HTML5 SDK that builds native-feeling mobile apps using web technologies like HTML, CSS, and Javascript. Because of those web technologies, it can run on a desktop browser which significantly simplifies testing. It is built on top of AngularJS to provide MVC architecture and Cordova for using native device functions with JavaScript code. Ionic is focused mainly on the look and feel and UI interaction of an app and simplifies one big part: the front end. The course comes with four starter projects to get students up and be able to start quickly.

The original version of this material can be found on project web page: `http://geniusgamedev.eu/teaching.php`.

Type: E-learning module, online-training, MOOC

Domain: Software development

Requirements: Basic knowledge in JavaScript programming in Cordova Framework

Target group: Computer Science students

License

Disclaimer

Ionic 3 basics

Ionic is a mobile framework built on top of AngularJS and Cordova, targeted at building multiplatform mobile apps. It uses the WebView component on iOS and Android to render it's content and implement functionality. The WebView is then wrapped in a native application.

Prerequisites

- nodeJS (Download)
- Cordova (`npm install -g cordova`)
- Ionic (`npm install -g ionic`)
- Android SDK or XCode (only Mac)

Important concepts

- Ionic uses the AngularJS MVC architecture
- HTML, TypeScript (Typescript in 30 minutes) and CSS are used to implement the UI
- Apache Cordova plugins give access to native device functions with JavaScript code
- Ionic CLI (command line interface). This is a NodeJS utility powered with commands for starting, building, running and emulating Ionic applications.

First project

When creating a project, there are a couple of options to choose from :

- Blank app
- Tabs app
- Side menu app

So if you want to create a new project with tabs :

```
ionic start myApp tabs
cd myApp
```

Adding a platform :

```
ionic cordova platform add android
```

Or :

```
ionic cordova platform add ios
```

Running your project in a browser :

```
ionic serve
```

Or in the Android emulator :

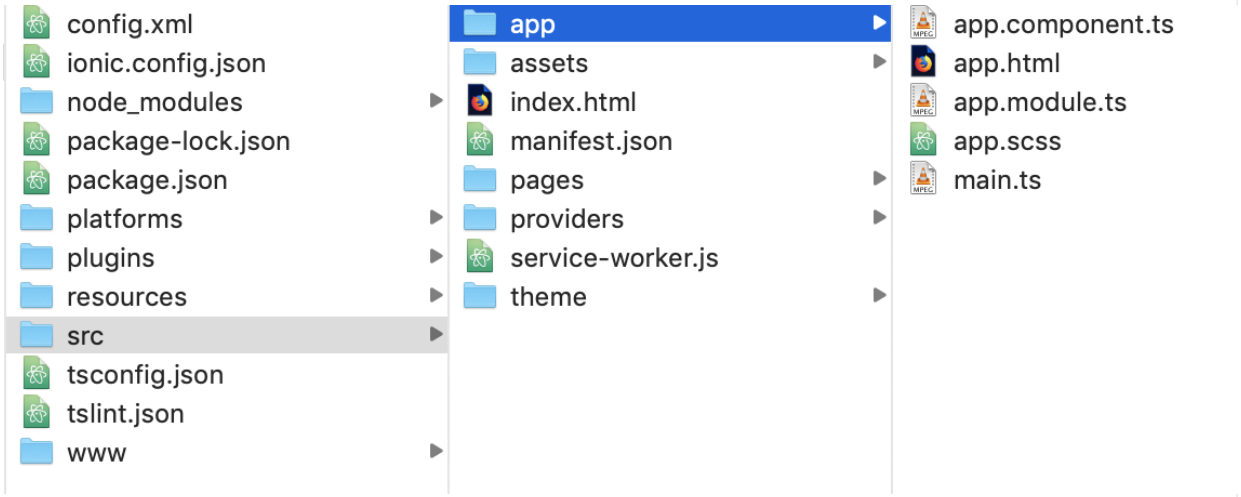
```
ionic cordova run android
```

Or in the iOS emulator :

```
ionic cordova run ios
```

Project Structure

The myApp project folder structure should look like this :



The most important folders and files are :

- node_modules : this contains all modules that Ionic needs to work. When uloading to github or compressing your project, you should empty this folder. Later on, you can restore it with `npm install`
- package.json : this is where all dependencies are managed (see npm install)
- platforms : this is the folder where Android and IOS projects are created. You might encounter some platform specific problems during development that will require these files, but you should leave them intact most of the time.
- plugins : this folder contains Cordova plugins. When you initially create an Ionic app, some plugins will be installed.
- resources : this folder is used for adding resources like icon and splash screen to your project.
- src : the main working folder for Ionic developers. When this folder is opened, you will find the following sub-folders :
  - app : this is where most of the work takes place
  - assets : fonts, icons and images
  - pages : the different screens in your project
  - theme : variables.scss is where you create a look and feel for your Ionic App by updating the $colors map and by overriding Ionic's Sass variables
- www : this is where the output of the build process is generated. It contains the runnable project. When this folder is opened, you will find the following sub-folders :
  - assets : fonts, icons and images
  - build : the generated CSS and JavaScript files
  - index.html as a starting point to your app. This contains the `<ion-app></ion-app>` tag

Within the src directory, these are the most important files :

- app.module.ts : the entry point for our app. Every app has a root module that essentially controls the rest of the application. This is where we declare all the pages in the app, import all modules, bootstrap our app using ionicBootstrap and declare all providers.

```
import { NgModule, ErrorHandler } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';

import { AboutPage } from '../pages/about/about';
import { ContactPage } from '../pages/contact/contact';
import { HomePage } from '../pages/home/home';
import { TabsPage } from '../pages/tabs/tabs';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

@NgModule({
  declarations: [
    MyApp,
    AboutPage,
    ContactPage,
    HomePage,
    TabsPage
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    AboutPage,
    ContactPage,
    HomePage,
    TabsPage
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler}
  ]
})
export class AppModule {}
```

- app.html : the main structure of the app

When the Ionic framework application is generated as a blank template it always comes with one existing page (component) HomePage. We can always add more pages with :

```
ionic generate page pageName
```

Each page comes with it's own html, ts and scss file. The .ts file is where the code for the class lives. For example, the contact page :

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
  selector: 'page-contact',
  templateUrl: 'contact.html'
})
export class ContactPage {

  constructor(public navCtrl: NavController) { }
}
```

Note how the NavController class is injected into the constructor of the ContactPage class. The NavController is used for navigating from and to other pages. You can learn more about it here.

Providers, Services and @Injectable

You can generate a new service with :

```
ionic generate provider myservice
```

This will generate a new service provider (MyserviceProvider) in the providers directory.
Providers, Services, and Injectables all reference the same thing, they are classes in our applications that are decorated with the @Injectable decorator :

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable()
export class MyserviceProvider {

  constructor(public http: HttpClient) {
    console.log('Hello MyserviceProvider Provider');
  }
}
```

set up as a provider in the app.module.ts file :

```
import { NgModule, ErrorHandler } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';

import { AboutPage } from '../pages/about/about';
import { ContactPage } from '../pages/contact/contact';
import { HomePage } from '../pages/home/home';
import { TabsPage } from '../pages/tabs/tabs';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';
import { MyserviceProvider } from '../providers/myservice/myservice';

@NgModule({
  declarations: [
    MyApp,
    AboutPage,
    ContactPage,
    HomePage,
    TabsPage
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    AboutPage,
    ContactPage,
    HomePage,
    TabsPage
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    MyserviceProvider
  ]
})
export class AppModule {}
```

and injected into classes that want to use them :

```
import { Component } from '@angular/core';
import { IonicPage } from 'ionic-angular';
import { MyserviceProvider } from '../../providers/myservice/myservice';

@IonicPage()
@Component({
  selector: 'page-something',
  templateUrl: 'something.html',
})
export class SomePage {
    constructor(public myService: MyserviceProvider) { }

    ionViewDidLoad() {
        this.myService.myFunction();
    }
}
```

One particular example of this is a data service. We can create a data service that handles saving and retrieving data for us. We could make a call to a service we have created to fetch some data for us without having to worry about what is happening behind the scenes.
In general just ask yourself 'is this task I am performing strictly related to this page and this page only?'' if the answer is yes, then you can probably just add the code directly to the page. If the answer is no, then you should probably create a service.

Navigation

Navigation in Ionic works like a simple stack, where we push new pages onto the top of the stack, which takes us forwards in the app. To go backwards, we pop the top page off. Using push to navigate to a new page is simple, and Ionic's navigation system is very flexible.

You don't have to implement a stack by yourself, Ionic 3 has already done that for you. You can easily accomplish navigation by using the NavController component which is injected into every page constructor alongside many other components.

The NavController component exposes many methods so you can control the navigation stack but NavController is not the only available component for navigation, there are other components:

- NavParams
- Nav
- NavBar

NavController

Let's say we have an About page and a Contact page. If we import those pages in our HomePage, then we can navigate to them by using the `push()` method :

```
import { AboutPage } from '../about/about';
import { ContactPage } from '../contact/contact';

 @Component({
   selector: 'page-home',
   templateUrl: 'home.html'
 })
 export class HomePage {

   constructor(public navCtrl: NavController) { }

   public gotoAbout(){
       this.navCtrl.push(AboutPage);
   }

   public gotoContact(){
       this.navCtrl.push(ContactPage);
   }
 }
```

Now these two methods can be made available on the home.html page :

```
<button ion-button (click)='gotoAbout();'>Go to about</button>
<button ion-button (click)='gotoContact()'>Go to contact</button>
```

To go back, we use the `pop()` method :

```
goBack(){
   this.navCtrl.pop();
}
```

NavParams

In many situations we need to pass parameters from one page to another for example from a list page to a detail page. You can pass parameters with the `push()` method of NavController as the second parameter of the method and retrieve them by injecting NavParams.

To pass them :

```
gotoAbout(){
  this.navCtrl.push(this.aboutPage,{param1 : "hello" , param2 : "world"});
}
```

And to retrieve them :

```
import { NavParams } from 'ionic-angular';

Injecting service and retrieving parameters

  @Component({
    selector: 'page-about',
    templateUrl: 'about.html'
  })
  export class AboutPage {
    private param1 : string ;
    private param2 : string ;
    private allParams ;

    constructor(public navCtrl: NavController, public navParams: NavParams) {
        this.param1 = this.navParams.get("param1");
        this.param2 = this.navParams.get("param2");
        this.allParams = this.navParams.data ;
    }
  }
```

Nav

Nav is a declarative equivalent of NavController which allows you to set the root page from HTML views. In src/app/app.html file, the view associated with the root app component, you find this code :

```
<ion-nav [root]='rootPage'></ion-nav>
```

In the src/app/app.component.ts file, the rootPage is set to HomePage :

```
@Component({
  templateUrl: 'app.html'
})
export class MyApp {
      rootPage = HomePage;
}
```

You can also set the root page using NavController' setRoot() method :

```
this.navCtrl.setRoot(HomePage);
```

Navbar

A Navbar is the navigational toolbar which you find on top of your app pages with a back button. A Navbar may contain a title, buttons, a segment and a search bar etc. In your homepage (home.html) you can find :

```
<ion-header>
<ion-navbar>
    <ion-title>
    Ionic Blank
    </ion-title>
</ion-navbar>
</ion-header>
```

Now you can add buttons to this Navbar :

```
<ion-header>
    <ion-navbar>
        <ion-title>
            Ionic 2 navigation demo
        </ion-title>
    </ion-navbar>
    <ion-buttons end>
        <button ion-button icon-only (click)='openModal()'>
            <ion-icon name="options"></ion-icon>
        </button>
    </ion-buttons>
</ion-header>
```

UI Components

Components

Ionic apps are made of high-level building blocks called components. Components allow you to quickly construct an interface for your app. Ionic comes with a number of components, including modals, popups, and cards. Check out the examples below to see what each component looks like and to learn how to use each one.

Action Sheets

Action Sheets slide up from the bottom edge of the device screen, and display a set of options with the ability to confirm or cancel an action. Action Sheets can sometimes be used as an alternative to menus, however, they **should not** be used for navigation.

The Action Sheet always appears above any other components on the page, and must be dismissed in order to interact with the underlying content. When it is triggered, the rest of the page darkens to give more focus to the Action Sheet options.

```
import { ActionSheetController } from 'ionic-angular';

export class MyPage {

  constructor(public actionSheetCtrl: ActionSheetController) { }

  presentActionSheet() {
    const actionSheet = this.actionSheetCtrl.create({
      title: 'Modify your album',
      buttons: [
        {
          text: 'Destructive',
          role: 'destructive',
          handler: () => {
            console.log('Destructive clicked');
          }
        },{
          text: 'Archive',
          handler: () => {
            console.log('Archive clicked');
          }
        },{
          text: 'Cancel',
          role: 'cancel',
          handler: () => {
            console.log('Cancel clicked');
          }
        }
      ]
    });
    actionSheet.present();
  }
}
```

Alerts

Alerts are a great way to offer the user the ability to choose a specific action or list of actions. They also can provide the user with important information, or require them to make a decision (or multiple decisions).

From a UI perspective, Alerts can be thought of as a type of "floating" modal that covers only a portion of the screen. This means Alerts should only be used for quick actions like password verification, small app notifications, or quick options. More in depth user flows should be reserved for full screen Modals.

Alerts are quite flexible, and can easily be customized.

- Basic Alerts
- Prompt Alerts
- Confirmation Alerts
- Radio Alerts
- Checkbox Alerts

Basic Alerts are typically used to notify the user about new information (a change in the app, a new feature, etc), an urgent situation that requires acknowledgement, or as a confirmation to the user that an action was successful or not.

```
import { AlertController } from 'ionic-angular';

export class MyPage {

  constructor(public alertCtrl: AlertController) { }

  showAlert() {
    const alert = this.alertCtrl.create({
      title: 'New Friend!',
      subTitle: 'Your friend, Obi wan Kenobi, just accepted your friend request!',
      buttons: ['OK']
    });
    alert.present();
  }
}
```

Prompt Alerts

Prompts offer a way to input data or information. Often times Prompts will be used to ask the user for a password before moving forward in an application's flow.

```
import { AlertController } from 'ionic-angular';

export class MyPage {

  constructor(public alertCtrl: AlertController) { }

  showPrompt() {
    const prompt = this.alertCtrl.create({
      title: 'Login',
      message: 'Enter a name for this new album you're so keen on adding',
      inputs: [
        {
          name: 'title',
          placeholder: 'Title'
        },
      ],
      buttons: [
        {
          text: 'Cancel',
          handler: data => {
            console.log('Cancel clicked');
          }
        },
        {
          text: 'Save',
          handler: data => {
            console.log('Saved clicked');
          }
        }
      ]
    });
    prompt.present();
  }
}
```

Confirmation Alerts

Confirmation Alerts are used when it is required that the user explicitly confirms a particular choice before progressing forward in the app. A common example of the Confirmation Alert is checking to make sure a user wants to delete or remove a contact from their address book.

```
import { AlertController } from 'ionic-angular';

export class MyPage {

  constructor(public alertCtrl: AlertController) { }

  showConfirm() {
    const confirm = this.alertCtrl.create({
      title: 'Use this lightsaber?',
      message: 'Do you agree to use this lightsaber to do good across the intergalactic galaxy?',
      buttons: [
        {
          text: 'Disagree',
          handler: () => {
            console.log('Disagree clicked');
          }
        },
        {
          text: 'Agree',
          handler: () => {
            console.log('Agree clicked');
          }
        }
      ]
    });
    confirm.present();
  }
}
```

Radio

Radio Alerts are a type of Confirmation Alert, but use the Radio component to offer several choices. A set of options is provided to the user, but only one option can be chosen.

```
import { AlertController } from 'ionic-angular';

export class MyPage {

  constructor(public alertCtrl: AlertController) { }

  showRadio() {
    let alert = this.alertCtrl.create();
    alert.setTitle('Lightsaber color');

    alert.addInput({
      type: 'radio',
      label: 'Blue',
      value: 'blue',
      checked: true
    });

    alert.addButton('Cancel');
    alert.addButton({
      text: 'OK',
      handler: data => {
        this.testRadioOpen = false;
        this.testRadioResult = data;
      }
    });
    alert.present();
  }
}
```

Checkbox

Checkbox Alerts are a type of Confirmation Alert, but use the Checkbox component to offer several choices. They offer the user a set of options to choose from.

```
import { AlertController } from 'ionic-angular';

export class MyPage {

  constructor(public alertCtrl: AlertController) { }

  showCheckbox() {
    let alert = this.alertCtrl.create();
    alert.setTitle('Which planets have you visited?');

    alert.addInput({
      type: 'checkbox',
      label: 'Alderaan',
      value: 'value1',
      checked: true
    });

    alert.addInput({
      type: 'checkbox',
      label: 'Bespin',
      value: 'value2'
    });

    alert.addButton('Cancel');
    alert.addButton({
      text: 'Okay',
      handler: data => {
        console.log('Checkbox data:', data);
        this.testCheckboxOpen = false;
        this.testCheckboxResult = data;
      }
    });
    alert.present();
  }
}
```

Badges

Badges are small components that typically communicate a numerical value to the user. They are typically used within an item.

```
<ion-item>
  <ion-icon name='logo-twitter' item-start></ion-icon>
  Followers
  <ion-badge item-end>260k</ion-badge>
</ion-item>
```

Badges can also be given any color attribute:

```
<ion-badge color='secondary'></ion-badge>
```

Buttons

Buttons are an essential way to interact with and navigate through an app, and should clearly communicate what action will occur after the user taps them. Buttons can consist of text and/or an icon, and can be enhanced with a wide variety of attributes.

For accessibility reasons, buttons use a standard `<button>` element, but are enhanced with an `ion-button` directive.

- Default Style
- Outline Style
- Clear Style
- Round Buttons
- Block Buttons
- Full Buttons
- Button Sizes
- Icon Buttons

- Buttons In Components

```
<button ion-button>Button</button>
```

The `color` property sets the color of the button. Ionic includes a number of default colors which can be easily overridden:

```
<button ion-button color='light'>Light</button>
<button ion-button>Default</button>
<button ion-button color='secondary'>Secondary</button>
<button ion-button color='danger'>Danger</button>
<button ion-button color='dark'>Dark</button>
```

Outline Style

To use the outline style for a button, just add the `outline` property:

```
<button ion-button color='light' outline>Light Outline</button>
<button ion-button outline>Primary Outline</button>
<button ion-button color='secondary' outline>Secondary Outline</button>
<button ion-button color='danger' outline>Danger Outline</button>
<button ion-button color='dark' outline>Dark Outline</button>
```

Clear Style

To use the clear style for a button, just add the `clear` property:

```
<button ion-button color='light' clear>Light Clear</button>
<button ion-button clear>Primary Clear</button>
<button ion-button color='secondary' clear>Secondary Clear</button>
<button ion-button color='danger' clear>Danger Clear</button>
<button ion-button color='dark' clear>Dark Clear</button>
```

Round Buttons

To create a button with rounded corners, just add the `round` property:

```
<button ion-button color='light' round>Light Round</button>
<button ion-button round>Primary Round</button>
<button ion-button color='secondary' round>Secondary Round</button>
<button ion-button color='danger' round>Danger Round</button>
<button ion-button color='dark' round>Dark Round</button>
```

Block Buttons

Adding `block` to a button will make the button take 100% of its parent's width. It will also add `display: block` to the button:

```
<button ion-button block>Block Button</button>
```

Full Buttons

Adding `full` to a button will also make the button take 100% of its parent's width. However, it will also remove the button's left and right borders. This style is useful when the button should stretch across the entire width of the display.

```
<button ion-button full>Full Button</button>
```

Button Sizes

Add the `large` attribute to make a button larger, or `small` to make it smaller:

```
<button ion-button small>Small</button>
<button ion-button>Default</button>
<button ion-button large>Large</button>
```

Icon Buttons

To add icons to a button, add an icon component inside of it and a position attribute:

```
<!-- Float the icon left -->
<button ion-button icon-start>
  <ion-icon name='home'></ion-icon>
  Left Icon
</button>

<!-- Float the icon right -->
<button ion-button icon-end>
  Right Icon
  <ion-icon name='home'></ion-icon>
</button>

<!-- Only icon (no text) -->
<button ion-button icon-only>
  <ion-icon name='home'></ion-icon>
</button>
```

Buttons In Components

Although buttons can be used on their own, they can easily be used within other components. For example, buttons can be added to a list item or a navbar.

```
<ion-header>
  <ion-navbar>
    <ion-buttons start>
      <button ion-button icon-only>
        <ion-icon name='contact'></ion-icon>
      </button>
    </ion-buttons>

    <ion-buttons end>
      <button ion-button icon-only>
        <ion-icon name='search'></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-list>
  <ion-item>
    Left Icon Button
    <button ion-button outline item-end icon-start>
      <ion-icon name='star'></ion-icon>
      Left Icon
    </button>
  </ion-item>
</ion-list>
```

Cards

Cards are a great way to display important pieces of content, and are quickly emerging as a core design pattern for apps. They are a great way to contain and organize information, while also setting up predictable expectations for the user. With so much content to display at once, and often so little screen realestate, cards have fast become the design pattern of choice for many companies, including the likes of Google, Twitter, and Spotify.

For mobile experiences, Cards make it easy to display the same information visually across many different screen sizes. They allow for more control, are flexible, and can even be animated. Cards are usually placed on top of one another, but they can also be used like a 'page' and swiped between, left and right.

- Basic Cards
- Card Headers
- Card Lists
- Card Images
- Background Images
- Advanced Cards

Cards are primarily a CSS component. To use a basic card, follow this structure:

```
<ion-card>

  <ion-card-header>
    Card Header
  </ion-card-header>

  <ion-card-content>
    <!-- Add card content here! -->
  </ion-card-content>

</ion-card>
```

Card Headers

Just like a normal page, cards can be customized to include headers. To add a header to a card, add the `<ion-card-header>` component inside of your card:

```
<ion-card>
  <ion-card-header>
    Header
  </ion-card-header>
  <ion-card-content>
    The British use the term 'header', but the American term 'head-shot' the English simply refuse to adopt.
  </ion-card-content>
</ion-card>
```

Lists In Cards

A card can contain a list of items. Add an `ion-list` component inside of the `ion-card-content` to display a list:

```
<ion-card>
  <ion-card-header>
    Explore Nearby
  </ion-card-header>

  <ion-list>
    <button ion-item>
      <ion-icon name='cart' item-start></ion-icon>
      Shopping
    </button>

    <button ion-item>
      <ion-icon name='medical' item-start></ion-icon>
      Hospital
    </button>

    <button ion-item>
      <ion-icon name='cafe' item-start></ion-icon>
      Cafe
    </button>

    <button ion-item>
      <ion-icon name='paw' item-start></ion-icon>
      Dog Park
    </button>

    <button ion-item>
      <ion-icon name='beer' item-start></ion-icon>
      Pub
    </button>

    <button ion-item>
      <ion-icon name='planet' item-start></ion-icon>
      Space
    </button>

  </ion-list>
</ion-card>
```

Images In Cards

Images often vary in size, so it is important that they adopt a consistent style throughout your app. Images can easily be added to cards. Adding an image to a card will give the image a constant width, and a variable height. Lists, headers, and other card components can easily be combined with image cards. To add an image to a card, use the following markup:

```
<ion-card>
  <img src='img/nin-live.png'/>
  <ion-card-content>
    <ion-card-title>
      Nine Inch Nails Live
      </ion-card-title>
    <p>
      The most popular industrial group ever, and largely
      responsible for bringing the music to a mass audience.
    </p>
  </ion-card-content>
</ion-card>
```

Background Images

Cards can be used to achieve a multitude of designs. We provide many of the elements to achieve common designs, but sometimes it will be necessary to add custom styles. Adding background images to cards is a perfect example of how adding custom styles can achieve a completely different look.

The following html can be added to the content of a page:

```
<ion-content class='card-background-page'>

  <ion-card>
    <img src='img/card-saopaolo.png'/>
    <div class='card-title'>São Paulo</div>
    <div class='card-subtitle'>41 Listings</div>
  </ion-card>

  <ion-card>
    <img src='img/card-amsterdam.png'/>
    <div class='card-title'>Amsterdam</div>
    <div class='card-subtitle'>64 Listings</div>
  </ion-card>

  <ion-card>
    <img src='img/card-sf.png'/>
    <div class='card-title'>San Francisco</div>
    <div class='card-subtitle'>72 Listings</div>
  </ion-card>

  <ion-card>
    <img src='img/card-madison.png'/>
    <div class='card-title'>Madison</div>
    <div class='card-subtitle'>28 Listings</div>
  </ion-card>

</ion-content>
```

Then, in the Sass file for the page:

```
.card-background-page {

  ion-card {
    position: relative;
    text-align: center;
  }

  .card-title {
    position: absolute;
    top: 36%;
    font-size: 2.0em;
    width: 100%;
    font-weight: bold;
    color: #fff;
  }

  .card-subtitle {
    font-size: 1.0em;
    position: absolute;
    top: 52%;
    width: 100%;
    color: #fff;
  }

}
```

Advanced Cards

The styles from different types of cards can be combined to create advanced cards. Cards can also use custom CSS. Below are a few advanced cards that have been built by combining various card attributes with a small amount of custom CSS.

- Social Cards
- Map Cards

Social Cards

It's often necessary to create social cards within an application. Using a combination of different items in a card you can achieve this.

```
<ion-card>

  <ion-item>
    <ion-avatar item-start>
      <img src='img/marty-avatar.png'>
    </ion-avatar>
    <h4>Marty McFly</h4>
    <p>November 5, 1955</p>
  </ion-item>

  <img src='img/advance-card-bttf.png'>

  <ion-card-content>
    <p>Wait a minute. Wait a minute, Doc. Uhhh... Are you telling me that you built a time machine... out of a DeLorean?!
Whoa. This is heavy.</p>
  </ion-card-content>

  <ion-row>
    <ion-col>
      <button ion-button icon-start clear small>
        <ion-icon name='thumbs-up'></ion-icon>
        <div>12 Likes</div>
      </button>
    </ion-col>
    <ion-col>
      <button ion-button icon-start clear small>
        <ion-icon name='text'></ion-icon>
        <div>4 Comments</div>
      </button>
    </ion-col>
    <ion-col center text-center>
      <ion-note>
        11h ago
      </ion-note>
    </ion-col>
  </ion-row>

</ion-card>
```

Map Cards

A combination of Ionic components can be used to create a card that appears as a map.

```
<ion-card>

  <img src='img/advance-card-map-madison.png'>
  <ion-fab right top>
    <button ion-fab>
      <ion-icon name='pin'></ion-icon>
    </button>
  </ion-fab>

  <ion-item>
    <ion-icon name='football' item-start large></ion-icon>
    <h4>Museum of Football</h4>
    <p>11 N. Way St, Madison, WI 53703</p>
  </ion-item>

  <ion-item>
    <ion-icon name='wine' item-start large ></ion-icon>
    <h4>Institute of Fine Cocktails</h4>
    <p>14 S. Hop Avenue, Madison, WI 53703</p>
  </ion-item>

  <ion-item>
    <span item-start>18 min</span>
    <span item-start>(2.6 mi)</span>
    <button ion-button icon-start clear item-end>
      <ion-icon name='navigate'></ion-icon>
      Start
    </button>
  </ion-item>

</ion-card>
```

Checkbox

A checkbox is an input component that holds a *boolean* value. Checkboxes are no different than HTML checkbox inputs. However, like other Ionic components, checkboxes are styled differently on each platform. Use the `checked` attribute to set the default value, and the `disabled` attribute to disable the user from changing the value.

```
<ion-item>
  <ion-label>Daenerys Targaryen</ion-label>
  <ion-checkbox color='dark' checked='true'></ion-checkbox>
</ion-item>

<ion-item>
  <ion-label>Arya Stark</ion-label>
  <ion-checkbox disabled='true'></ion-checkbox>
</ion-item>
```

DateTime

The DateTime component is used to present an interface which makes it easy for users to select dates and times. The DateTime component is similar to the native `<input type='datetime-local'>` element, however, Ionic's DateTime component makes it easy to display the date and time in a preferred format, and manage the datetime values.

```
<ion-item>
  <ion-label>Start Time</ion-label>
  <ion-datetime displayFormat='h:mm A' pickerFormat='h mm A' [(ngModel)]='event.timeStarts'></ion-datetime>
</ion-item>
```

FABs

FABs (Floating Action Buttons) are standard material design components. They are shaped as a circle that represents a promoted action. When pressed, it may contain more related actions. FABs as its name suggests are floating over the content in a fixed position.

```
<ion-content>
  <!-- Real floating action button, fixed. It will not scroll with the content -->
  <ion-fab top right edge>
    <button ion-fab mini><ion-icon name='add'></ion-icon></button>
    <ion-fab-list>
      <button ion-fab><ion-icon name='logo-facebook'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-twitter'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-vimeo'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-googleplus'></ion-icon></button>
    </ion-fab-list>
  </ion-fab>

  <ion-fab right bottom>
    <button ion-fab color='light'><ion-icon name='arrow-dropleft'></ion-icon></button>
    <ion-fab-list side='left'>
      <button ion-fab><ion-icon name='logo-facebook'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-twitter'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-vimeo'></ion-icon></button>
      <button ion-fab><ion-icon name='logo-googleplus'></ion-icon></button>
    </ion-fab-list>
  </ion-fab>

</ion-content>
```

Gestures

Basic gestures can be accessed from HTML by binding to `tap`, `press`, `pan`, `swipe`, `rotate`, and `pinch` events.

```
<ion-card (tap)='tapEvent($event)'>
  <ion-item>
    Tapped: {{tap}} times
  </ion-item>
</ion-card>
```

Grid

Ionic's grid system is based on flexbox, a CSS feature supported by all devices that Ionic supports. The grid is composed of three units — *grid*, *rows* and *columns*.

The grid system is made of a 12 columns, and each `ion-col` can be sized by setting the `col-<width>` attribute.

```
<ion-grid>
  <ion-row>
    <ion-col col-12>This column will take 12 columns</ion-col>
  </ion-row>
  <ion-row>
    <ion-col col-6>This column will take 6 columns</ion-col>
  </ion-row>
</ion-grid>
```

An `ion-col` can be sized for different breakpoints by setting the `col-<breakpoint>-<width>`.

```
<ion-grid>
  <ion-row>
    <ion-col col-12 col-sm-9 col-md-6 col-lg-4 col-xl-3>
      This column will be 12 columns wide by default,
      9 columns at the small breakpoint,
      6 at the medium breakpoint, 4 at large, and 3 at xl.
    </ion-col>
  </ion-row>
</ion-grid>
```

Icons

Ionic comes with around 700+ icons.

To use an icon, populate the `name` attribute on the `ion-icon` component:

```
<ion-icon name='heart'></ion-icon>
```

Active / Inactive Icons

All icons have both `active` and `inactive` states. Active icons are typically full and thick, where as inactive icons are outlined and thin. Set the `isActive` attribute to `true` or `false` to change the state of the icon. Icons will default to active if a value is not specified.

```
<ion-icon name='heart'></ion-icon>                  <!-- active -->
<ion-icon name='heart' isActive='false'></ion-icon>  <!-- inactive -->
```

Platform Specific Icons

Many icons have both **Material Design** and **iOS** versions. Ionic will automatically use the correct version based on the platform.

To specify the icon to use for each platform, use the `md` and `ios` attributes and provide the platform specific icon name.

```
<ion-icon ios='logo-apple' md='logo-android'></ion-icon>
```

Variable Icons

To set an icon using a variable:

```
<ion-icon [name]='myIcon'></ion-icon>
```

```
export class MyFirstPage {
  // use the home icon
  myIcon: string = 'home';
}
```

Inputs

Inputs are essential for collecting and handling user input in a secure way. They should follow styling and interaction guidelines for each platform, so that they are intuitive for users to interact with. Ionic uses Angular 2's form library, which can be thought of as two dependent pieces, **Controls**, and **Control Groups**.

Each input field in a form has a **Control**, a function that binds to the value in the field, and performs validation. A **Control Group** is a collection of Controls. Control Groups handle form submission, and provide a high level API that can be used to determine whether the entire form is valid.

A number of attributes that can be used to style forms and their various input fields are listed below.

- Fixed Inline Labels
- Floating Labels
- Inline Labels
- Inset Labels
- Placeholder Labels
- Stacked Labels

Fixed Inline Labels

Use `fixed` to place a label to the left of the input element. The label does not hide when text is entered. The input will align on the same position, regardless of the length of the label. Placeholder text can be used in conjunction with a fixed label.

```
<ion-list>

  <ion-item>
    <ion-label fixed>Username</ion-label>
    <ion-input type='text' value=''></ion-input>
  </ion-item>

  <ion-item>
    <ion-label fixed>Password</ion-label>
    <ion-input type='password'></ion-input>
  </ion-item>

</ion-list>
```

Floating Labels

Floating labels are labels that animate or "float" up when the input is selected. Add the `floating` attribute to an `ion-label` to use.

Enter text in the example to the right to see the floating labels in action.

```
<ion-list>

  <ion-item>
    <ion-label floating>Username</ion-label>
    <ion-input type='text'></ion-input>
  </ion-item>

  <ion-item>
    <ion-label floating>Password</ion-label>
    <ion-input type='password'></ion-input>
  </ion-item>

</ion-list>
```

Inline Labels

An `ion-label` without any attributes is an inline label. The label does not hide when text is entered. Placeholder text can be used in conjunction with an inline label.

```
<ion-list>

  <ion-item>
    <ion-label>Username</ion-label>
    <ion-input type='text'></ion-input>
  </ion-item>

  <ion-item>
    <ion-label>Password</ion-label>
    <ion-input type='password'></ion-input>
  </ion-item>

</ion-list>

<div padding>
  <button block>Sign In</button>
</div>
```

Inset Labels

By default each input item will fill 100% of the width of its parent element. Make the input inset by adding the `inset` attribute to the `ion-list` component.

```
<ion-list inset>

  <ion-item>
    <ion-label>Username</ion-label>
    <ion-input type='text'></ion-input>
  </ion-item>

  <ion-item>
    <ion-label>Password</ion-label>
    <ion-input type='password'></ion-input>
  </ion-item>

</ion-list>
```

Placeholder Labels

Add the `placeholder` attribute to an `<input>` element to simulate the input's label. When text is entered into the input, the placeholder label is hidden.

```
<ion-list>

  <ion-item>
    <ion-input type='text' placeholder='Username'></ion-input>
  </ion-item>

  <ion-item>
    <ion-input type='password' placeholder='Password'></ion-input>
  </ion-item>

</ion-list>
```

Stacked Labels

A stacked label will always appear on top of the input. Add the `stacked` attribute to an `ion-label` to use. Placeholder text can be used in conjunction with an stacked label.

```
<ion-list>

  <ion-item>
    <ion-label stacked>Username</ion-label>
    <ion-input type='text'></ion-input>
  </ion-item>

  <ion-item>
    <ion-label stacked>Password</ion-label>
    <ion-input type='password'></ion-input>
  </ion-item>

</ion-list>
```

Lists

Lists are used to display rows of information, such as a contact list, playlist, or menu.

- Basic Lists
- Inset List
- List Dividers
- List Headers
- Icon List
- Avatar List
- Multi-line List
- Sliding List
- Thumbnail List

By default, all lists will be styled with divider lines:

```
<ion-list>
  <button ion-item *ngFor='let item of items' (click)='itemSelected(item)'>
    {{ item }}
  </button>
</ion-list>
```

No Lines

Adding the `no-lines` attribute will hide the dividers between list items:

```
<ion-list no-lines>
  <button ion-item *ngFor='let item of items' (click)='itemSelected(item)'>
    {{ item }}
  </button>
</ion-list>
```

Inset List

Lists don't have an outside margin by default. To add one, add the `inset` attribute to `ion-list` component.

```
<ion-list inset>
  <button ion-item *ngFor='let item of items' (click)='itemSelected(item)'>
    {{ item }}
  </button>
</ion-list>
```

List Dividers

To divide groups of items, use `<ion-item-group>` instead of `<ion-list>`. Use `<ion-item-divider>` components to divide the group in to multiple sections:

```
<ion-content>
  <ion-item-group>
    <ion-item-divider color='light'>A</ion-item-divider>
    <ion-item>Angola</ion-item>
    <ion-item>Argentina</ion-item>
  </ion-item-group>
</ion-content>
```

List Headers

Each list can include a header at the top of the list:

```
<ion-list>
  <ion-list-header>
    Action
  </ion-list-header>
  <ion-item>Terminator II</ion-item>
  <ion-item>The Empire Strikes Back</ion-item>
  <ion-item>Blade Runner</ion-item>
</ion-list>
```

Icon List

Adding icons to list items is a great way to hint about the contents of each item. The position of the icon can be set using the `item-start` and `item-end` attributes. The size of the icon defaults to `small`, and can be made larger with the `large` attribute.

```
<ion-list>
  <ion-item>
    <ion-icon name='leaf' item-start></ion-icon>
      Herbology
    <ion-icon name='rose' item-end></ion-icon>
  </ion-item>
</ion-list>
```

Avatar List

Item avatars showcase an image larger than an icon, but smaller than a thumbnail. To use an avatar, add an `<ion-avatar>` component inside of an item. The position of the avatar can be set using the `item-start` and `item-end` attributes:

```
<ion-list>
  <ion-item>
    <ion-avatar item-start>
      <img src='img/avatar-cher.png'>
    </ion-avatar>
    <h4>Cher</h4>
    <p>Ugh. As if.</p>
  </ion-item>
</ion-list>
```

Multi-line List

Multi-line lists are identical to regular lists, except items have multiple lines of text. When an `<ion-item>` component contains multiple header or paragraph elements, it will automatically expand it's height to fit the new lines of text. Below is an example with three lines of text:

```
<ion-list>
  <ion-item>
    <ion-avatar item-start>
      <img src='img/avatar-finn.png'>
    </ion-avatar>
    <h4>Finn</h4>
    <h3>Don't Know What To Do!</h3>
    <p>I've had a pretty messed up day. If we just...</p>
  </ion-item>
</ion-list>
```

Sliding List

Sliding items can be swiped to the left or right to reveal a hidden set of buttons. To use a sliding item, add an `ion-item-sliding` component inside of an `ion-list` component. Next, add an `<ion-item-options>` component inside of the sliding item to contain the buttons.

```
<ion-list>
  <ion-item-sliding>
    <ion-item>
      <ion-avatar item-start>
        <img src='img/slimer.png'>
      </ion-avatar>
      <h4>Slimer</h4>
    </ion-item>
    <ion-item-options side='left'>
      <button ion-button color='primary'>
        <ion-icon name='text'></ion-icon>
        Text
      </button>
      <button ion-button color='secondary'>
        <ion-icon name='call'></ion-icon>
        Call
      </button>
    </ion-item-options>
    <ion-item-options side='right'>
      <button ion-button color='primary'>
        <ion-icon name='mail'></ion-icon>
        Email
      </button>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

Thumbnail List

Item thumbnails showcase an image that takes up the entire height of an item. To use a thumbnail, add an `<ion-thumbnail>` component inside of an item. The position of the thumbnail can be set using the `item-start` and `item-end` attributes:

```
<ion-list>
  <ion-item>
    <ion-thumbnail item-start>
      <img src='img/thumbnail-totoro.png'>
    </ion-thumbnail>
    <h4>My Neighbor Totoro</h4>
    <p>Hayao Miyazaki • 1988</p>
    <button ion-button clear item-end>View</button>
  </ion-item>
</ion-list>
```

Loading

The Loading component is an overlay that prevents user interaction while indicating activity. By default, it shows a spinner based on the mode. Dynamic content can be passed and displayed with the spinner. The spinner can be hidden or customized to use several predefined options. The loading indicator is presented on top of other content even during navigation.

```
import { LoadingController } from 'ionic-angular';

export class MyPage {

  constructor(public loadingCtrl: LoadingController) { }

  presentLoading() {
    const loader = this.loadingCtrl.create({
      content: 'Please wait...',
      duration: 3000
    });
    loader.present();
  }
}
```

Menus

Menu is a side-menu navigation that can be dragged out or toggled to show. The content of a menu will be hidden when the menu is closed.

Menu adapts to the appropriate style based on the platform.

```
<ion-menu [content]='content'>
  <ion-header>
    <ion-toolbar>
      <ion-title>Menu</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    <ion-list>
      <button ion-item (click)='openPage(homePage)'>
        Home
      </button>
      <button ion-item (click)='openPage(friendsPage)'>
        Friends
      </button>
      <button ion-item (click)='openPage(eventsPage)'>
        Events
      </button>
      <button ion-item (click)='closeMenu()'>
        Close Menu
      </button>
    </ion-list>
  </ion-content>
</ion-menu>

<ion-nav id='nav' #content [root]='rootPage'></ion-nav>
```

Modals

Modals slide in off screen to display a temporary UI, often used for login or signup pages, message composition, and option selection.

```
import { ModalController } from 'ionic-angular';
import { ModalPage } from './modal-page';

export class MyPage {

  constructor(public modalCtrl: ModalController) { }

  presentModal() {
    const modal = this.modalCtrl.create(ModalPage);
    modal.present();
  }
}
```

Navigation

Navigation is how users move between different pages in your app. Ionic's navigation follows standard native navigation concepts, like those in iOS. In order to enable native-like navigation, we've built a few new navigation components that might feel different for developers used to traditional desktop browser navigation.

There are several ways to navigate throughout an Ionic app:

Basic Navigation

Navigation is handled through the `<ion-nav>` component, which works as a simple stack that new pages are pushed onto and popped off of, corresponding to moving forward and backward in history.

We start with a root page that loads the Nav component:

```
import { StartPage } from 'start'

@Component({
  template: '<ion-nav [root]='rootPage'></ion-nav>'
})
class MyApp {
  // First page to push onto the stack
  rootPage = StartPage;
}
```

Next, we can access the Navigation Controller in each page that is navigated to by injecting it into any of our Pages. Note that Page components does not need a selector. Ionic adds these automatically .

```
@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Login</ion-title>
    </ion-navbar>
  </ion-header>

  <ion-content>Hello World</ion-content>`
})
export class StartPage {

  constructor(public navCtrl: NavController) { }

}
```

To navigate from one page to another simply push or pop a new page onto the stack:

```
@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Login</ion-title>
    </ion-navbar>
  </ion-header>

  <ion-content>
    <button (click)='goToOtherPage()'>
      Go to OtherPage
    </button>
  </ion-content>`
})
export class StartPage {

  constructor(public navCtrl: NavController) { }

  goToOtherPage() {
    //push another page onto the history stack
    //causing the nav controller to animate the new page in
    this.navCtrl.push(OtherPage);
  }
}

@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Other Page</ion-title>
    </ion-navbar>
  </ion-header>

  <ion-content>I'm the other page!</ion-content>`
})
class OtherPage { }
```

If your page has an `<ion-navbar>`, a back button will automatically be added to it if it is not a root page, and the title of the Nav Bar will be updated.

Alternatively, if you want to go back, but don't have a NavBar, you can pop the current page off the stack:

```
@Component({
  template: `
  <ion-content>
    <button (click)='goBack()'>
      There's no place like home
    </button>
  </ion-content>`
})
class OtherPage {

  constructor(public navCtrl: NavController) { }

  goBack() {
    this.navCtrl.pop();
  }
}
```

Navigating from the Root Component

What if you want to control navigation from your root app component? You can't inject `NavController` because any components that are navigation controllers are *children* of the root component so they aren't available to be injected.

By adding a reference variable to the `ion-nav`, you can use `@ViewChild` to get an instance of the `Nav` component, which is a navigation controller (it extends `NavController`):

```
@Component({
  template: '<ion-nav #myNav [root]='rootPage'></ion-nav>'
})
export class MyApp {
  @ViewChild('myNav') nav;
  rootPage = TabsPage;

  // Wait for the components in MyApp's template to be initialized
  // In this case, we are waiting for the Nav identified by
  // the template reference variable #myNav
  ngAfterViewInit() {
    // Let's navigate from TabsPage to Page1
    this.nav.push(LoginPage);
  }
}
```

Tabbed Navigation

Navigation can be nested and used inside of complex components like Tabs. Unlike traditional routing systems, Ionic's navigation makes it easy to navigate to a given Page from anywhere in the app without specifying a specific route to it. To use the App Store app on iOS as an example, we can easily navigate to the AppDetailPage that shows info about a specific app from any tab (try it yourself to see!). Take a look at the Tabs docs for more info on how to easily achieve this.

Popover

The Popover is a view that floats above an app's content. Popovers provide an easy way to present or gather information from the user and are commonly used in the following situations:

- Show more info about the current view
- Select a commonly used tool or configuration
- Present a list of actions to perform inside one of your views

```
import { PopoverController } from 'ionic-angular';
import { MyPopOverPage } from './my-pop-over';

export class MyPage {

  constructor(public popoverCtrl: PopoverController) { }

  presentPopover() {
    const popover = this.popoverCtrl.create(MyPopOverPage);
    popover.present();
  }
}
```

Radio

Like the checkbox, a radio is an input component that holds a *boolean* value. Under the hood, radios are no different than HTML radio inputs. However, like other Ionic components, radios are styled differently on each platform. Unlike checkboxes, radio components form a group, where only one radio can be selected at a time. Use the `checked` attribute to set the default value, and the `disabled` attribute to disable the user from changing to that value.

```
<ion-list radio-group>
  <ion-list-header>
    Language
  </ion-list-header>

  <ion-item>
    <ion-label>Go</ion-label>
    <ion-radio checked='true' value='go'></ion-radio>
  </ion-item>

  <ion-item>
    <ion-label>Rust</ion-label>
    <ion-radio value='rust'></ion-radio>
  </ion-item>

  <ion-item>
    <ion-label>Python</ion-label>
    <ion-radio value='python' disabled='true'></ion-radio>
  </ion-item>
</ion-list>
```

Range

A Range is a control that lets users select from a range of values by moving a slider knob along the slider bar or track.

```
<ion-item>
  <ion-range [(ngModel)]='brightness'>
    <ion-icon range-left small name='sunny'></ion-icon>
    <ion-icon range-right name='sunny'></ion-icon>
  </ion-range>
</ion-item>
```

Searchbar

A Searchbar binds to a model, and emits an input event when the model is changed.

```
<ion-searchbar (ionInput)='getItems($event)'></ion-searchbar>
<ion-list>
  <ion-item *ngFor='let item of items'>
    {{ item }}
  </ion-item>
</ion-list>
```

Note that in this example, the `getItems()` function is called when the input changes, which updates the cities that are displayed. Although this example filters the list based on the search input, Searchbar can be used in many different scenarios:

```
@Component({
  templateUrl: 'search/template.html',
})
class SearchPage {

  searchQuery: string = '';
  items: string[];

  constructor() {
    this.initializeItems();
  }

  initializeItems() {
    this.items = [
      'Amsterdam',
      'Bogota',
      ...
    ];
  }

  getItems(ev: any) {
    // Reset items back to all of the items
    this.initializeItems();

    // set val to the value of the searchbar
    const val = ev.target.value;

    // if the value is an empty string don't filter the items
    if (val && val.trim() != '') {
      this.items = this.items.filter((item) => {
        return (item.toLowerCase().indexOf(val.toLowerCase()) > -1);
      })
    }
  }
}
```

Segment

Segment is a collection of buttons that are displayed in line. They can act as a filter, showing/hiding elements based on the segments value.

```
<div padding>
  <ion-segment [(ngModel)]='pet'>
    <ion-segment-button value='kittens'>
      Kittens
    </ion-segment-button>
    <ion-segment-button value='puppies'>
      Puppies
    </ion-segment-button>
  </ion-segment>
</div>

<div [ngSwitch]='pet'>
  <ion-list *ngSwitchCase=''puppies''>
    <ion-item>
      <ion-thumbnail item-start>
        <img src='img/thumbnail-puppy-1.jpg'>
      </ion-thumbnail>
      <h4>Ruby</h4>
    </ion-item>
    ...
  </ion-list>

  <ion-list *ngSwitchCase=''kittens''>
    <ion-item>
      <ion-thumbnail item-start>
        <img src='img/thumbnail-kitten-1.jpg'>
      </ion-thumbnail>
      <h4>Luna</h4>
    </ion-item>
    ...
  </ion-list>
</div>
```

Select

The `ion-select` component is similar to an HTML `<select>` element, however, Ionic's select component makes it easier for users to sort through and select the preferred option. When users tap the select component, a dialog will appear with all of the options in a large, easy to select list.

```
<ion-list>
  <ion-item>
    <ion-label>Gaming</ion-label>
    <ion-select [(ngModel)]='gaming'>
      <ion-option value='nes'>NES</ion-option>
      <ion-option value='n64'>Nintendo64</ion-option>
      <ion-option value='ps'>PlayStation</ion-option>
      <ion-option value='genesis'>Sega Genesis</ion-option>
      <ion-option value='saturn'>Sega Saturn</ion-option>
      <ion-option value='snes'>SNES</ion-option>
    </ion-select>
  </ion-item>
</ion-list>
```

multiple selections can be made with `<ion-select>` by adding `multiple='true'` to the component.

```
<ion-list>
  <ion-item>
    <ion-label>Toppings</ion-label>
    <ion-select [(ngModel)]='toppings' multiple='true' cancelText='Nah' okText='Okay!'>
      <ion-option value='bacon' selected='true'>Bacon</ion-option>
      <ion-option value='olives'>Black Olives</ion-option>
      <ion-option value='xcheese' selected='true'>Extra Cheese</ion-option>
      <ion-option value='peppers'>Green Peppers</ion-option>
      <ion-option value='mushrooms'>Mushrooms</ion-option>
      <ion-option value='onions'>Onions</ion-option>
      <ion-option value='pepperoni'>Pepperoni</ion-option>
      <ion-option value='pineapple'>Pineapple</ion-option>
      <ion-option value='sausage'>Sausage</ion-option>
      <ion-option value='Spinach'>Spinach</ion-option>
    </ion-select>
  </ion-item>
</ion-list>
```

Slides

Slides make it easy to create galleries, tutorials, and page-based layouts. Slides take a number of configuration options on the `<ion-slides>` component.

```
<ion-slides pager>

  <ion-slide style='background-color: green'>
    <h4>Slide 1</h4>
  </ion-slide>

  <ion-slide style='background-color: blue'>
    <h4>Slide 2</h4>
  </ion-slide>

  <ion-slide style='background-color: red'>
    <h4>Slide 3</h4>
  </ion-slide>

</ion-slides>
```

Tabs

Tabs powers a multi-tabbed interface with a Tab Bar and a set of views that can be tabbed through.

- Text Tabs
- Icon Tabs
- Text and Icon Tabs
- Badge Tabs

To initialize Tabs, use `<ion-tabs>`, with a child `<ion-tab>` for each tab:

```
import { Component } from '@angular/core';
import { Tab1 } from './tab1-page';
import { Tab2 } from './tab2-page';

@Component({
  template: `
    <ion-tabs>
      <ion-tab tabIcon='heart' [root]='tab1'></ion-tab>
      <ion-tab tabIcon='star' [root]='tab2'></ion-tab>
    </ion-tabs>`
})
class MyApp {

  tab1: any;
  tab2: any;

  constructor() {
    this.tab1 = Tab1;
    this.tab2 = Tab2;
  }
}
```

Individual tabs are just `@Components`

```
import { Component } from '@angular/core';

@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Heart</ion-title>
    </ion-navbar>
  </ion-header>
  <ion-content>Tab 1</ion-content>`
})
export class Tab1 { }

@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Star</ion-title>
    </ion-navbar>
  </ion-header>
  <ion-content>Tab 2</ion-content>`
})
export class Tab2 { }
```

Notice that each `<ion-tab>` binds to a `[root]` property, just like `<ion-nav>` in the Navigation section above. That is because each `<ion-tab>` is really just a navigation controller. This means that each tab has its own history stack, and `NavController instances injected` into children `@Components` of each tab will be unique to each tab:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
...
})
class Tab1 {

  constructor(public navCtrl: NavController) {
    // Id is 1, nav refers to Tab1
    console.log(this.nav.id)
  }
}

@Component({
...
})
class Tab2 {

  constructor(public navCtrl: NavController) {
    // Id is 2, nav refers to Tab2
    console.log(this.nav.id)
  }
}
```

Icon Tabs

To add an icon inside of a tab, use the `tab-icon` attribute. This attribute can be passed the name of any icon:

```
@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Tabs</ion-title>
    </ion-navbar>
  </ion-header>
  <ion-content></ion-content>
  `
})
class TabContentPage {

  constructor() { }

}

@Component({
  template: `
  <ion-tabs>
    <ion-tab tabIcon='contact' [root]='tab1'></ion-tab>
    <ion-tab tabIcon='compass' [root]='tab2'></ion-tab>
    <ion-tab tabIcon='analytics' [root]='tab3'></ion-tab>
    <ion-tab tabIcon='settings' [root]='tab4'></ion-tab>
  </ion-tabs>`
})
export class TabsIconPage {

  constructor() {
    this.tab1 = TabContentPage;
    this.tab2 = TabContentPage;
    ...
  }

}
```

Icon and Text

To add text and an icon inside of a tab, use the `tab-icon` and `tab-title` attributes:

```
@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Tabs</ion-title>
    </ion-navbar>
  </ion-header>
  <ion-content></ion-content>
  `
})
class TabsTextContentPage {

  constructor() { }

}

@Component({
  template: `
  <ion-tabs>
    <ion-tab tabIcon='water' tabTitle='Water' [root]='tab1'></ion-tab>
    <ion-tab tabIcon='leaf' tabTitle='Life' [root]='tab2'></ion-tab>
    <ion-tab tabIcon='flame' tabTitle='Fire' [root]='tab3'></ion-tab>
    <ion-tab tabIcon='magnet' tabTitle='Force' [root]='tab4'></ion-tab>
  </ion-tabs>`
})
export class TabsTextPage {

  constructor() {
    this.tab1 = TabsTextContentPage;
    this.tab2 = TabsTextContentPage;
    ...
  }

}
```

Badges

To add a badge to a tab, use the `tabBadge` and `tabBadgeStyle` attributes. The `tabBadgeStyle` attribute can be passed the name of any color:

```
@Component({
  template: `
  <ion-header>
    <ion-navbar>
      <ion-title>Tabs</ion-title>
    </ion-navbar>
  </ion-header>
  <ion-content></ion-content>
  `
})
class TabBadgePage {

  constructor() { }

}

@Component({
  template: `
    <ion-tabs>
      <ion-tab tabIcon='call' [root]='tabOne' tabBadge='3' tabBadgeStyle='danger'></ion-tab>
      <ion-tab tabIcon='chatbubbles' [root]='tabTwo' tabBadge='14' tabBadgeStyle='danger'></ion-tab>
      <ion-tab tabIcon='musical-notes' [root]='tabThree'></ion-tab>
    </ion-tabs>
    `
})
export class BadgesPage {

  constructor() {
    this.tabOne = TabBadgePage;
    this.tabTwo = TabBadgePage;
  }

}
```

Toast

Toast is a subtle notification that appears on top of an app's content. Typically, Toasts are displayed for a short duration of time then automatically dismiss.

```
import { ToastController } from 'ionic-angular';

export class MyPage {

  constructor(public toastCtrl: ToastController) { }

  presentToast() {
    const toast = this.toastCtrl.create({
      message: 'User was added successfully',
      duration: 3000
    });
    toast.present();
  }
}
```

Toggle

A toggle is an input component that holds a *boolean* value. Like the checkbox, toggles are often used to allow the user to switch a setting on or off. Attributes like `value`, `disabled`, and `checked` can be applied to the toggle to control its behavior.

```
<ion-item>
  <ion-label>Sam</ion-label>
  <ion-toggle disabled checked='false'></ion-toggle>
</ion-item>
```

Toolbar

A toolbar is a generic bar that can be used in an app as a header, sub-header, footer, or even sub-footer. Since `ion-toolbar` is based on flexbox, no matter how many toolbars you have in your page, they will be displayed correctly and `ion-content` will adjust accordingly.

Note: Typically a NavBar is used inside the `ion-header` when used in conjunction with navigation.

- Changing the Color
- Buttons in Toolbars
- Segment in Toolbars
- Searchbar in Toolbars

```
<ion-header>
  <ion-toolbar>
    <ion-title>Toolbar</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content></ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-title>Footer</ion-title>
  </ion-toolbar>
</ion-footer>
```

Header

One of the best uses of the toolbar is as a header.

```
<ion-header>
  <ion-toolbar color='primary'>
    <ion-buttons start>
      <button ion-button icon-only>
        <ion-icon name='more'></ion-icon>
      </button>
    </ion-buttons>

    <ion-title>Header</ion-title>

    <ion-buttons end>
      <button ion-button icon-only>
        <ion-icon name='search'></ion-icon>
      </button>
    </ion-buttons>

  </ion-toolbar>
</ion-header>

<ion-content>
  <p>There is a header above me!</p>
</ion-content>
```

Changing the Color

You can change the toolbars color by changing the property on the element.

```
@Component({
  template: `
    <ion-toolbar color='primary'>
      <ion-title>Toolbar</ion-title>
    </ion-toolbar>


    <ion-toolbar color='secondary'>
      <ion-title>Toolbar</ion-title>
    </ion-toolbar>

    <ion-toolbar color='danger'>
      <ion-title>Toolbar</ion-title>
    </ion-toolbar>


    <ion-toolbar color='dark'>
      <ion-title>Toolbar</ion-title>
    </ion-toolbar>
  `
  })
```

You can also change the navbar's color the same way. This will allow you to have a different color navbar per page in your app.

```
<ion-header>
  <ion-navbar color='dark'>
    <ion-title>Dark</ion-title>
  </ion-navbar>
</ion-header>


<ion-header>
  <ion-navbar color='danger'>
    <ion-title>Danger</ion-title>
  </ion-navbar>
</ion-header>

<ion-header>
  <ion-navbar color='secondary'>
    <ion-title>Secondary</ion-title>
  </ion-navbar>
</ion-header>
```

Buttons in Toolbars

Buttons can be added to both header and footer toolbars. To add a button to a toolbar, we need to first add an `ion-buttons` component. This component wraps one or more buttons, and can be given the `start` or `end` attributes to control the placement of the buttons it contains:

```
<ion-header>
  <ion-toolbar>
    <ion-buttons start>
      <button ion-button icon-only color='royal'>
        <ion-icon name='search'></ion-icon>
      </button>
    </ion-buttons>
    <ion-title>Send To...</ion-title>
    <ion-buttons end>
      <button ion-button icon-only color='royal'>
        <ion-icon name='person-add'></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>

<ion-content></ion-content>

<ion-footer>
  <ion-toolbar>
    <p>Ash, Misty, Brock</p>
    <ion-buttons end>
      <button ion-button icon-end color='royal'>
        Send
        <ion-icon name='send'></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-footer>
```

Segment in Toolbars

Segments are a great way to allow users to switch between different sets of data. Use the following markup to add a segment to a toolbar:

```
<ion-header>
  <ion-toolbar>
    <ion-buttons start>
      <button ion-button icon-only>
        <ion-icon name='create'></ion-icon>
      </button>
    </ion-buttons>
    <ion-segment>
      <ion-segment-button value='new'>
        New
      </ion-segment-button>
      <ion-segment-button value='hot'>
        Hot
      </ion-segment-button>
    </ion-segment>
    <ion-buttons end>
      <button ion-button icon-only>
        <ion-icon name='more'></ion-icon>
      </button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

Searchbar in Toolbars

Another common design paradigm is to include a searchbar inside your toolbar.

```
<ion-header>
  <ion-toolbar color='primary'>
    <ion-searchbar (input)='getItems($event)'></ion-searchbar>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-item *ngFor='let item of items'>
      {{ item }}
    </ion-item>
  </ion-list>
</ion-content>
```

Starter Projects

Following starter projects are provided in this tutorial :

- localstorage example
- location example
- quiz example
- pouchdb example

CouchDB and Ionic 3

Let's start by creating a new Ionic 3 project, using the Ionic CLI. So go ahead and open the terminal on Mac/Linux or the command prompt in Windows the run the following command to generate a new project.

```
ionic start ionic3-pouchdb-sqlite blank
```

Wait for the project to setup the dependencies then navigate inside the root folder:

```
cd ionic3-pouchdb-sqlite
```

Installing SQLite Cordova Plugin and PouchDB

On Ionic 3, the native SQLite database is the most adequate choice when it comes to storing data locally, because it allows to have unlimited storage which is not the case of local storage or IndexedDB. Also SQLite is a file based database which makes very portable.

You can add SQLite support to your Ionic application with many Cordova plugins such as:

- Cordova-sqlite-storage: the original Cordova plugin for SQLite
- cordova-plugin-sqlite-2: a fork of the previous plugin with extra features

Let's install the fork with :

```
ionic cordova plugin add cordova-plugin-sqlite-2
```

To be able to use PouchDB, you need to install a third party library available from npm :

```
npm install pouchdb --save
```

Next you'll need to install another third part adapter to use SQLite with PouchDB :

```
npm install pouchdb-adapter-cordova-sqlite --save
```

These are all the dependencies you need to install in order to start using PouchDB with SQLite in your Ionic app. We will build a small application that allows to do CRUD operations i.e create, read, update and delete employees data from a PouchDB + SQLite database. The application has many screens:

- employees screen: list employees with delete buttons
- create an employee screen
- update an employee screen

Services

The first thing we need to create is the database service that connects to PouchDB and provides different methods to work with the employees database. We can create a service provider using ionic g provider

Make sure you are inside the project's root folder then run the following:

```
ionic g provider employee
```

You should have an EmployeeProvider provider generated inside src/providers/employee folder.
Now replace the code in employee.ts with :

```
import cordovaSqlitePlugin from 'pouchdb-adapter-cordova-sqlite';
import { Injectable } from '@angular/core';
import 'rxjs/add/operator/map';
import PouchDB from 'pouchdb';

@Injectable()
export class EmployeeProvider {

  public pdb;
  employees;

  createPouchDB() {
    PouchDB.plugin(cordovaSqlitePlugin);
    this.pdb = new PouchDB('employee.db', {
      adapter: 'cordova-sqlite',
      iosDatabaseLocation: 'Library',
      androidDatabaseImplementation: 2
    });
  }
}
```

This creates a SQLite database file named employees.db and initializes the PouchDB database by setting the adapter to cordova-sqlite which instructs PouchDB to use SQLite for storage instead of browser's storage. Make sure to import the provider into src/app/app.module.ts and add it to the providers array if it's not added automatically. Also make sure to import the service provider and inject it in the constructor of any component before you can use it.
Now create the CRUD methods in EmployeeProvider.
First the create method :

```
create(employee) {
  return this.pdb.post(employee);
}
```

`post()` is a PouchDB API that allows you to create new objects in the PouchDB database.
Now the update method :

```
update(employee) {
  return this.pdb.put(employee);
}
```

Next the delete method :

```
delete(employee) {
  return this.pdb.remove(employee);
}
```

And finally the read method :

```
read() {
  let pdb = this.pdb;

  function allDocs() {

    let _employees = pdb.allDocs({ include_docs: true })
      .then(docs => {
        return docs.rows;
      });

    return Promise.resolve(_employees);
  };
  return allDocs();
}
```

This gets all employees from the database by invoking the `allDocs()` method which returns a promise that resolves to an array of all employees in the database. The `map()` function then maps the array to rows which contain the documents itself (and not other PouchDB specific data). The code also converts row.doc.Date (stored as JSON) to JavaScript Date().

Screens

Let's create different pages that allows us to list and do actions on the employees database. We already have a home page generated for us which lives in src/app/pages/home. Open home.html then update the code to show the list of employees using `<ion-list>`

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Ionic Employee Database
    </ion-title>
    <ion-buttons end>
      <button ion-button (click)="addEmployee()">
        <ion-icon name="add"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list>
    <ion-item *ngFor="let emp of employees" (click)="showDetails(emp)">
      <div>
        <h3>{{emp.doc.firstName}} {{emp.doc.lastName}}</h3>
      </div>
    </ion-item>
  </ion-list>
</ion-content>
```

Next we need to add the code to get employees in src/pages/home/home.ts so open the file then update it to match with the following code:

```
import { Component } from '@angular/core';
import { IonicPage, NavController, ModalController } from 'ionic-angular';
import { EmployeeProvider } from '../../../providers/employee/employee';

@IonicPage()
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  public employees;

  constructor(
    public navCtrl: NavController,
    public modalCtrl: ModalController,
    public empProv: EmployeeProvider
  ) {}

  ionViewDidEnter() {
    this.empProv.createPouchDB();

    this.empProv.read()
      .then(employees => {
        this.employees = employees;
      }).catch((err) => { console.log(err)} );
  }

  showDetails(employee) {
    let modal = this.modalCtrl.create('EmployeePage', { employee: employee });
    modal.onDidDismiss(data => {
      this.reReadEmployees();
    });
    modal.present();
  }

  addEmployee() {
    let modal = this.modalCtrl.create('EmployeePage', { employee: null });
    modal.onDidDismiss(data => {
      this.reReadEmployees()
    });
    modal.present();
  }

  reReadEmployees() {
    this.empProv.read()
      .then(employees => {
        this.employees = employees;
      }).catch((err) => { console.log(err)} );
  }
}
```

Next we need to generate a new page for employee details so run the following command in your terminal:

```
ionic g page employee
```

This will generate an employee folder inside app/pages/employee with three files. Open employee.html then update it to match the following code:

```
<ion-header>
  <ion-navbar>
    <ion-title>Employee Details</ion-title>
    <ion-buttons end *ngIf="canDelete">
      <button ion-button (click)="delete()">
        <ion-icon name="trash"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-item>
      <ion-label>First Name</ion-label>
      <ion-input text-right type="text" [(ngModel)]="employee.firstName"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Last Name</ion-label>
      <ion-input text-right type="text" [(ngModel)]="employee.lastName"></ion-input>
    </ion-item>
  </ion-list>
  <button ion-button block (click)="addOrUpdate()">Add/Update Employee</button>
</ion-content>
```

Next update src/pages/employee/employee.ts to add the required logic code

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams, ViewController } from 'ionic-angular';
import { EmployeeProvider } from '../../../providers/employee/employee';

@IonicPage()
@Component({
  selector: 'page-employee',
  templateUrl: 'employee.html',
})
export class EmployeePage {
  employee: any = {};
  canDelete = false;
  canUpdate = false;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    private employeeProvider: EmployeeProvider,
    public viewCtrl: ViewController
  ) {}

  ionViewDidEnter() {
    var employee = this.navParams.get('employee');
    if (employee) {
      this.employee = employee.doc;
      this.canDelete = true;
      this.canUpdate = true;
    }
  }

  addOrUpdate() {
    if (this.canUpdate) {
      this.employeeProvider.update(this.employee).catch(() => {});
    }
    else {
      this.employeeProvider.create(this.employee).catch(() => {});
    }
    this.viewCtrl.dismiss(this.employee);
  }

  delete() {
    this.employeeProvider.delete(this.employee).catch(() => {});
    this.viewCtrl.dismiss(this.employee);
  }
}
```

Replication

By default, we are going to run into CORS (Cross Origin Resource Sharing) issues when trying to interact with CouchDB. You may get an error like this:

```
XMLHttpRequest cannot load http://localhost:5984/employee/?_nonce=1466856096255. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:8100' is therefore not allowed access.
```

To fix this, you can simply install the add-cors-to-couchdb package.
Run the following command:

```
npm install -g add-cors-to-couchdb
```

Then run the following command with CouchDB up and running :

```
add-cors-to-couchdb
```

If it worked, you should get a message saying "Success". This will configure CouchDB correctly, and you only ever need to run this once.
Now add following code to `createPouchDB()` in EmployeeProvider :

```
this.remote = 'http://localhost:5984/employees';

let options = {
  live: true,
  retry: true,
  continuous: true
};

this.pdb.sync(this.remote, options);
```

This sets up live (or continuous) bi-directional syncronisation of the PouchDB with CoudchBD.
If you need more options or other configurations, have a look at : https://pouchdb.com/guides/replication.html

For the fully working example, click here