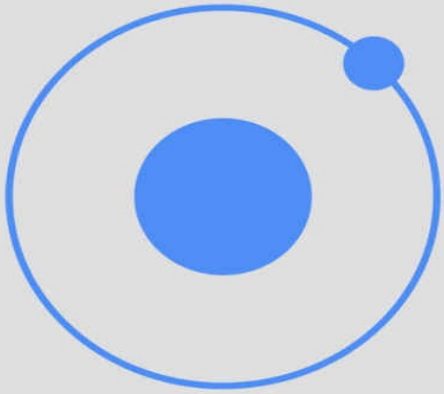
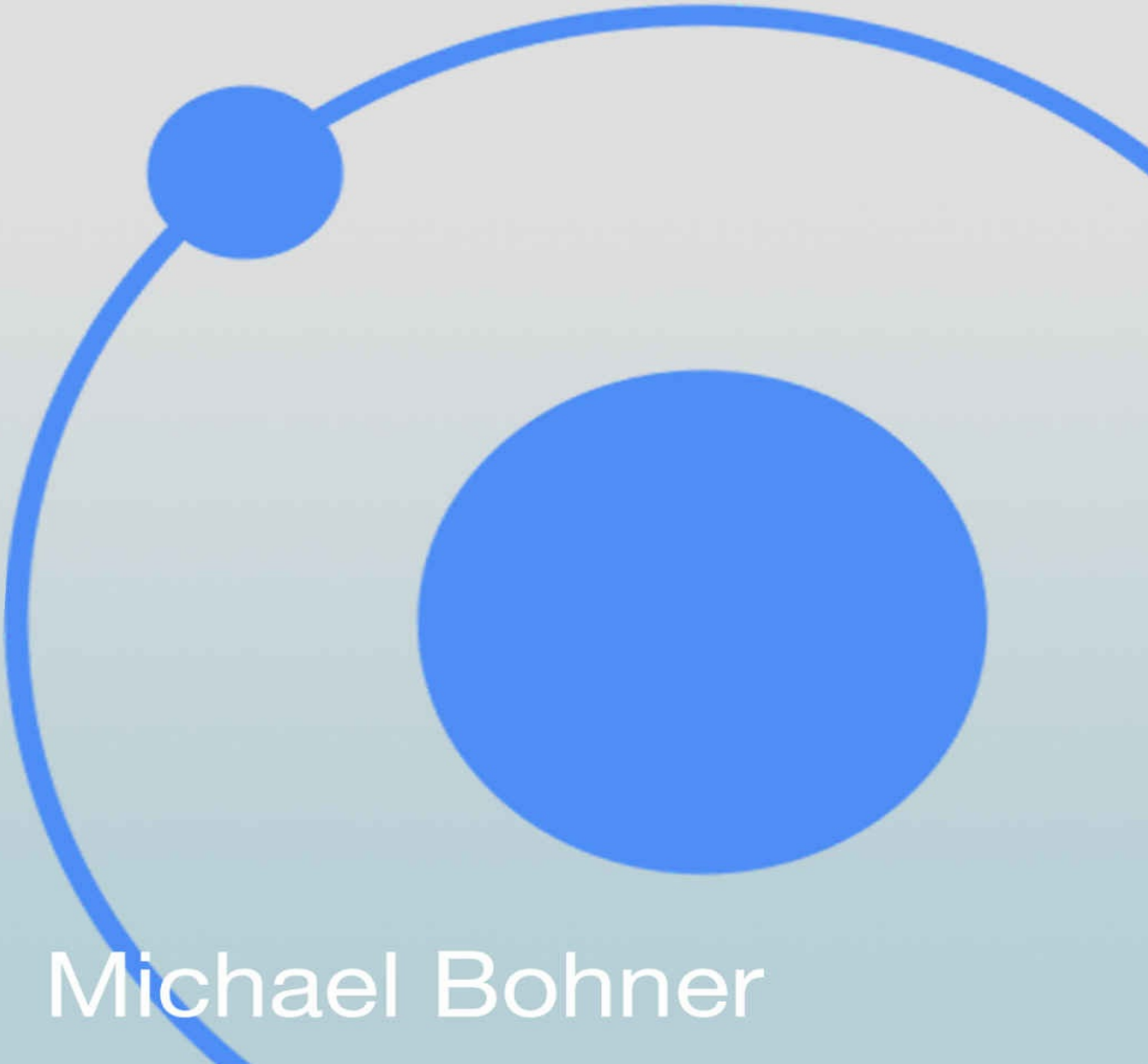


ionic

Building mobile apps with Ionic Framework



Michael Bohner

Ionic Framework

Building mobile apps with Ionic Framework

By Michael Bohner

Table of contents

Disclaimer

While all attempts have been made to verify the information provided in this book, the author does assume any responsibility for errors, omissions, or contrary interpretations of the subject matter contained within. The information provided in this book is for educational and entertainment purposes only. The reader is responsible for his or her own actions and the author does not accept any responsibilities for any liabilities or damages, real or perceived, resulting from the use of this information.

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Introduction

The need for mobile apps is on the rise. This is because of the increased use of mobile devices for both Android and iOS. The Ionic Framework can be used for development of these apps. Its apps can be used on Android 4+ and iOS 6+. For one to use Ionic for mobile app development, you do not have to learn the native programming languages such as Java and Swift. This is why you should learn to use this framework. This book will guide you in this.

Chapter 1- Definition

Ionic is just an HTML5 framework for mobile app development. Note that the mobile apps developed with Ionic are hybrid. This means that they are small websites which run in the browser shell in your application which is capable of accessing the native platform layer. The framework makes it possible for web developers to develop mobile applications without the need for them to learn the native programming languages such as the Swift, Objective-C, and Java. The Ionic SDK is an open-source project, meaning that you can download and use it for free on your system. The framework was built on top of the Apache Cordova and the AngularJS. In this framework, web technologies such as the CSS, HTML5, and Sass are used for the purpose of developing hybrid mobile apps.

Hybrid apps have numerous benefits associated with them compared to the purely native apps. With these apps, we are capable of accessing third party code, the speed of development is very high, and numerous platforms can be supported. You should think of it as the front-end User Interface (UI) which determines the look and feel of the app and provides for interaction with the users. It can be compared to Bootstrap, but it supports a broad range of mobile components including beautiful design and slick animations. This book will guide you in learning how to develop mobile apps by the use of Ionic.

Chapter 2- Installation

Before beginning to develop your mobile apps, you should begin by setting up the environment ready for the Ionic Framework. You should download the Ionic Framework and then install any of its dependencies which are necessary. Currently, the framework is targeting Android and iPhone devices. iOS 6+ an Android 4+ are supported by the Ionic Framework. The development of mobile apps by use of Ionic Framework can be done in any of the operating systems that one wants to use. However, if you are developing apps for your iOS, then you should use the Mac OS X.

For Windows users, you should begin by downloading and installing *"Git for Windows"* and *"Console2."* These are the ones which you will use for the execution of your commands. We will begin by installing the latest version of Apache Cordova, and this will be responsible for taking the app and then bundling it into the native wrapper so as to turn it into a traditional native app. However, before the installation of Apache Cordova is done, one should begin by installing the *"NodeJS."* The following command can be used for installation of the Apache Cordova:

```
$ sudo npm install -g cordova
```

If the software is being installed on Windows, then you can do away with the *"sudo"* command. Note that there are some tools which are specific to a particular platform, so make sure that you adhere to this depending on the Operating System (OS) that you are using on your system.

Installation of Ionic

Now that you have the Apache Cordova installed, we can go ahead to install the Ionic Framework. This will provide us with a command line utility which will help us to start the building and packaging of the Ionic apps. This framework can be installed by running the following command:

```
$ sudo npm install -g ionic
```

Creating the Project

We now need to start writing the code for our app. We should then create a Cordova project which will provide us with the environment for this. Just run the following command:


```
$ sudo npm install -g ionic
```


After execution of the above command, a folder named "*todo*" will be created in your system. You need to be clear on the directory in which the above command is executed, as it is where the folder will be created. The outer structure of the Ionic project will be as

follows:

├── bower.json // dependencies for bower

├── config.xml // configuration for cordova

├── gulpfile.js // gulp tasks

├── hooks // custom cordova hooks for execution on specific commands

├── ionic.project // configuration for ionic

├── package.json // dependencies for node

├── platforms // iOS/Android specific builds will be available here

├── plugins // the plugins for cordova/ionic plugins will be installed here

├── scss // the scss code, for outputting to www/css/

└── www // application - JS code and libs, CSS, images, etc.

If you need to view the above structure of your project, then just execute the following command:

```
$ cd todo && ls
```

For those of you who need to make use of a version control system, then you can proceed with setting up its folder in this location.

Configuring the Platforms

In this step, our aim is to alert the Ionic Framework that we need to perform the enabling of the Android and iOS platforms. For those who are not using OS X, then you can do away with the iOS platform. Just run the following commands on your system:

```
$ ionic platform add ios
$ ionic platform add android
```

For some of you, errors might arise if the above commands are executed. If this happens, just make sure that you install all of the platform tools which are needed by following the steps given above.

Once you are done this, it is good for you to verify whether everything is good. This can be done by building and then running the project. To do this, just execute the commands given below:

```
$ ionic build ios
$ ionic emulate ios
```

Note that for those who are building the project for Android, substitute the "*ios*" in the

above commands with “*android.”* If you find that everything is ready, then all the better, and you will be set to start building your app.

Chapter 3- How to Start the Node Server

The Node server should be responsible for exposing of the conference data, which includes the speakers and sessions, and this will be done through a set of REST services. To install this, the following steps can be followed:

1. Begin by cloning the repository or by downloading the support files for the project. The cloning can be done by executing the following command:

```
git clone https://github.com/ccoenraets/ionic-tutorial
```

   Once the zip has been downloaded, you should extract it in a separate directory from the one in which it has been downloaded to.

2. For Mac users, open the terminal window, while for Windows users, open the command window. You can then use the "*cd"command* so as to navigate to the directory "*ionic-tutorial/server."*

3. Use the following command so as to install the dependencies for the server:

```
npm install
```

4. Start the server by running the following command:

```
node server
```

   Some of you might get an error after running the above command. If this happens, then make sure that there is no other server which is listening to port number 5000.

5. After that, the REST services can then be tested. This can be done by opening the browser and then accessing the following URLs:
   - http://localhost:5000/sessions- this will open the list of conference sessions which are returned as a JSON format.
   - http://localhost:5000/sessions/1- this will give you information about a specific session.

Chapter 4- Creating a Mockup using Iconic Creator

Now that the environment is already set up, we can begin the process of building our mobile apps. In this chapter, we need to use the Iconic Creator so as to create the views for our app.

The User Interface

With the Ionic Framework, there are UI components which one can use so as to create their app. Once can use CSS so as to customize these. For you to get started with creating your app, you have two choices which you can follow. These include the following:

- Create from a template
- Create a mockup in Ionic Creator

Using a Template

The template provides you with an easy and faster way for you to create your apps. There are numerous templates which are available in Ionic including maps, side menu, tabs, tests, blank, sales-force, and complex-list.

If you need to create the project files for Ionic for your app, just run the following command:

```
$ ionic start [appName] [templateName]
```

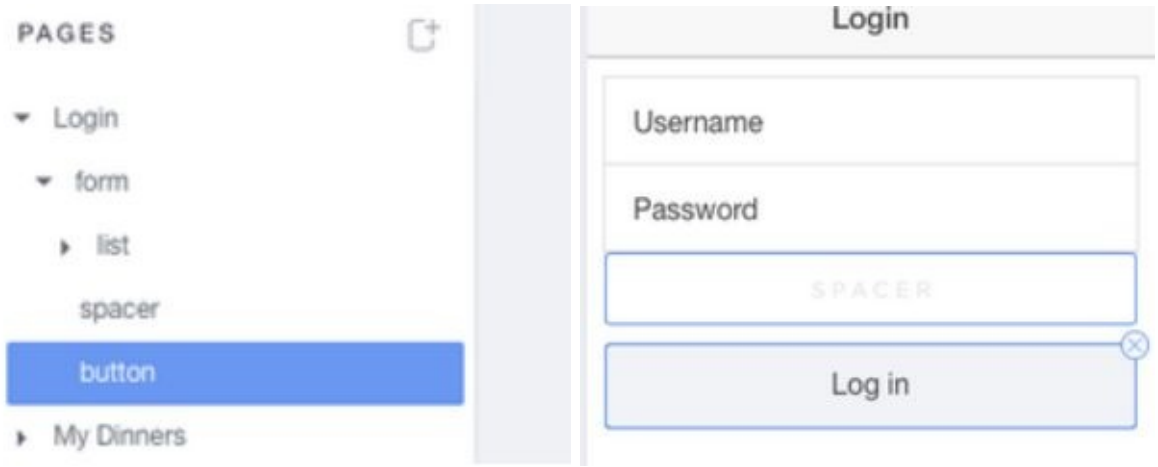
Note that you should choose both the "*appName"* and the "*templateName."*

Using the Ionic Creator

This is just a web application, and it lets you create the views for your app by use of the drag-and-drop interface. Once the views have been created, they can be exported to the code, and then used in the app.

You should begin by signing up for the Iconic Creator. Once you have provided a name for your project, you will just be logged into your account. To demonstrate how this can be used, we will guide you on how to create a login screen for your app.

On the top-right corner of the left corner, identify the icon labeled "+." Click on it and all of the available templates will be presented to you. Select on the template for login.



You can then add a new and blank page. The properties in the right pane should then be changed. From the pane for "*COMPONENTS,"* you can add a card component. Open the login view, and then click on the "*Login"* button. The property "*LINK"* located on the right side should then be changed.

Exporting the Code

The Ionic Creator provides three ways how the code can be exported. These include the following:

- Ionic CLI
- Raw HTML
- ZIP file

In this chapter, we will use the Ionic CLI for exporting the code and creating the Ionic project for our app at once. From the header, click on the "*Export"* button and then choose "*Ionic CLI."* Identify the second line which looks as follows, and then copy it:

ionic start [appName] creator:1539943175db

Just open the Terminal window, and then execute the line which you have copied. This will create an Ionic project for you. The "*appName"* should be substituted for the name for the app which you need. You should now have a folder which contains all of the Ionic files for your project. Navigate to the "*www"* folder, identify the "*index.html"* file, and then open it in your text editor. A routing code will be generated, and it will be as follows:

.config(function($stateProvider, $urlRouterProvider) {

$stateProvider

.state('p1', {

url: '/login',

templateUrl: ‘p1.html’

})

.state(‘p2’, {

url: ‘/our-dinners’,

templateUrl: ‘p2.html’

})

;

// the following should be used as the fallback if none of the above is //matched

$urlRouterProvider.otherwise(‘/login’);

});

For Ionic to transition from one view to another, it uses the “*AngularUI Router*.” As shown in

From the above code, once you click on the “*Login”* button, you will be moved to “*p1,”* while if you click on the other URL, you will be taken to the page “*p2.”*

The generated HTML code for the view will be as follows:

<body ng-app=“app” animation=“slide-left-right-ios7”>

** **

** **

<ion-nav-bar class=“bar-stable”>

<ion-nav-back-button class=“button-icon icon ion-ios7-arrow-back”>Back</ion-nav-back-button>

</ion-nav-bar>

```
<ion-nav-view></ion-nav-view>
</div>
</div>
<script id="page1.html" type="text/ng-template">
<ion-view title="Login">
<ion-content padding="true" scroll="false" class="has-header">
<form>
<ion-list>
<label class="item item-input">
<span class="input-label">Your Name</span>
<input type="text" placeholder="">
</label>
<label class="item item-input">
<span class="input-label">Your Password</span>
<input type="password" placeholder="">
</label>
</ion-list>
<div class="spacer" style="width: 310px; height: 19px;"></div>
<a href="#/my-dinners" class="button button-light button-block">Log in</a>
</form>
</ion-content>
</ion-view></script>
<script id="p2.html" type="text/ng-template">
<ion-view title="Our Dinners">
<ion-content padding="true" class="has-header">
<div class="list card">
<div class="item item-divider">Our Dinner</div>
```

<div class=“item item-body”>

** **

** **

**<i class=“icon ion-calendar” style=“margin-right: 5px”></i>August 26, 2015 **

** **

**<i class=“icon ion-location” style=“margin-right: 11px”></i>Our Region **

** **

** **

** **

</ion-content>

</ion-view></script>

</body>

Fixing of the Styles

You have noticed that there are some iOS specific styles in the code which has been generated. Our aim is to remove this styling, since the app might look weird when run on an Android device. This is given below:

<body ng-app=“application”>

** **

** **

<ion-nav-bar class=“bar-stable”>

<ion-nav-back-button></ion-nav-back-button>

</ion-nav-bar>

A style sheet named “*preview-frame.css”* has also been included in the code, and this should be removed. It is shown below:

<link href=”/css/preview-frame.css” rel=“stylesheet”>

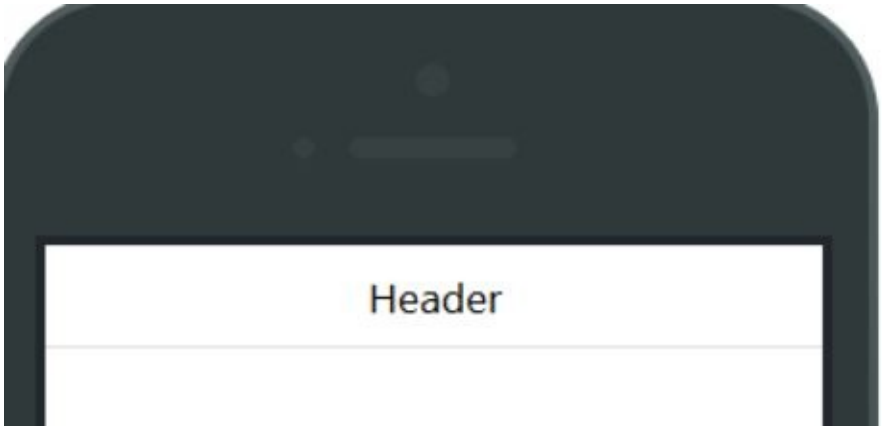
You will then be done with this step.

Chapter 5- Ionic Framework Components

The Ionic Framework has numerous components which are used for the process of developing the mobile apps. These will be discussed in this chapter.

Header

The header is the label at the top of the app which can be used for writing the title. And the right and the left buttons which can be used for navigation or for carrying out some specified actions.



There are different color options for headers. Examples of these are given below:

bar-light:

** **

<h1 class="title">A light bar</h1>

** **

This just gives you a colorless bar.

bar-positive:

The code for this should be as follows:

<div class=“bar bar-header bar-positive”>

<h1 class=“title”>The bar-positive</h1>

** **

This gives you the following bar:

bar-balanced:

The code for this should be as follows:

<div class=“bar bar-header bar-calm”>

<h1 class=“title”>The bar-calm</h1>

** **

The above code should give you a bar similar to the following:

bar-assertive:

This kind of bar takes the following code:

<div class=“bar bar-header bar-assertive”>

<h1 class=“title”>A bar-assertive</h1>

** **

The code will give you a bar similar to the one given below:

Buttons

As far as mobile apps are concerned, buttons are very essential. However, the appearance of a button will determine how your attractive your mobile app is to the users. Ionic Framework comes with various types of buttons differing in size, color, and other features. Let us discuss some of these buttons:

button-positive:

The code for this button should be as follows:

<button class="button button-positive">

button-positive

</button>

The button should be as follows:

button-light:

This kind of button uses the following code:

<button class="button button-light">

button-light

</button>

The code should give you the following button:

button-assertive:

The code for creating this button should be as follows:

<button class="button button-assertive">

button-assertive

</button>

The code should give you the following kind of a button:

button-royal:

The following should be the code for this kind of button:

<button class="button button-royal">

button-royal

</button>

The code should give you the following kind of button:

There exist several other types of buttons. However, you notice that the syntax for creating them is very easy to learn. To enhance the appearance of your app, make sure that you are able to create the different types of buttons.

Icon Buttons

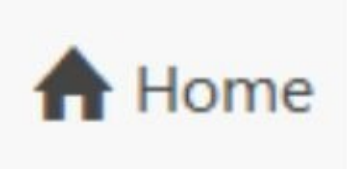
Buttons having icons make it easy for users, and especially the novice ones to find it easy to use your apps. Some people are greatly attracted to graphics rather than the text. This explains the need for icon buttons. Ionic Framework supports the use of these. However, when you add an icon to your button, then the number of elements which can be held is also reduced. Let us give examples of some buttons having icons in them:

Home button:

This can be implemented by use of the following code:

<button class=“button icon-left ion-home”>Home</button>

The code should give you the following button:



The icon for “*home”* is present in the above button.

Settings button:

The code for implementing this button should be as follows:

****

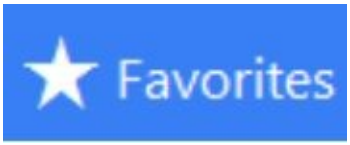
The code will give you a button similar to the one given below:

Favorites button:

This button can be created by use of the following code:

<button class = “button icon-left ion-star button-positive”> Favorites </button>

The code should give you a button with an icon as shown below:



Back button:

This button should be implemented by use of the following code:

Back

The result for the above code should be as follows:

Forms and Inputs

A “*list*” can be used for the grouping together of related items.

Placeholders

It is good for us to guide users when they are providing information via forms. With labels, this is made possible. However, when the user begins to type in the text field having the label, then the placeholder will be hidden. Consider the example given below:

** **

<label class=“item item-input”>

<input type=“text” placeholder=“First Name”>

</label>

<label class=“item item-input”>

<input type=“text” placeholder=“Last Name”>

</label>

** **

On executing the above code, the following output should be observed:

With the above interface, the user will easily see the kind of data which is required in which field.

Inline Labels

The class "*input-label*" can be used for placing of a label inside your input label. However, the label will not be hidden when the user begins to type the text. Consider the example given below which shows how this can be implemented:

<label class="item item-input">

Username

<input type="text">

</label>

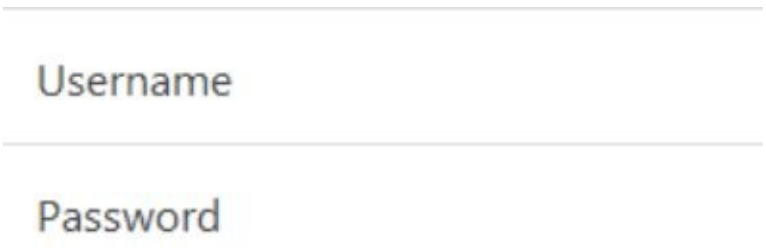
<label class="item item-input">

Password

<input type="password">

</label>

The above code should give you the following interface:



Floating Labels

With these, once one has entered some text in the input field, the label will float or animate. Consider the example given below showing how these can be implemented:

<label class=“item item-input item-floating-label”>

Your First Name

<input type=“text” placeholder=“First Name”>

</label>

<label class=“item item-input item-floating-label”>

Your Last Name

<input type=“text” placeholder=“Last Name”>

</label>

<label class=“item item-input item-floating-label”>

Your Email

<input type=“text” placeholder=“Email”>

</label>

The above code should give output similar to the one given below:

First Name

Last Name

Email

Inset Forms

The default setting in Ionic Framework is that each of the inputs which you use should fill the parent 100%. However, you have the choice of changing this setting. Consider the example given below which shows how an inset form can be created in Ionic Framework:

** **

<label class="item item-input">

<input type="text" placeholder="First Name">

</label>

<label class="item item-input">

<input type="text" placeholder="Last Name">

</label>

** **

The above code should give you the following form when executed:

As shown in the above form, the form does not reach the width of the device. Note that we have used the class "*list-inset.*". That is how simple this kind of form can be implemented in Ionic Framework.

Inset Inputs

Consider a situation in which we need to implement a particular input field together with a button in the same item. This can be done by use of the "*inset input.*" The following code can be used for that purpose:

** **

<label class="item-input-wrapper">

<input type="text" placeholder="Email">

</label>

<button class="button button-large">

Submit

</button>

** **

The above code should give you output similar to the one given below:

That is how simply the elements can be implemented within the same item.

Input icons

We said that icons are very useful, particularly in guiding users who are attracted not by text but by graphics. Consider the example given below which shows how icons can be implemented in input elements:

** **

<label class="item item-input">

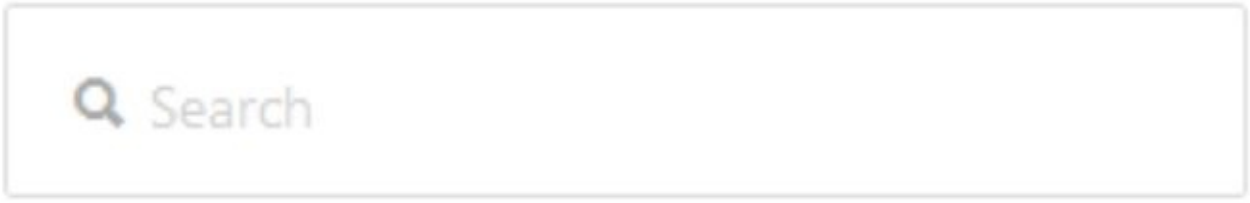
****

<input type="text" placeholder="Search">

</label>

** **

The above code should give you the following output:



Toggle

The toggle button is similar to the checkbox in HTML, with the exception in how it looks and that users find it easy for them to use it on a device with a touch screen. In Ionic, this element is wrapped within the "*<label>*" element, and this makes it easy for us to tap and drag it.

The toggle can take different colors depending on the type that we use. Consider the code given below:

<label class="toggle">

<input type="checkbox">

** **

** **

** **

</label>

That is how a toggle button can be created in Ionic Framework.

Checkbox

In Ionic Framework, the checkbox is not different from the checkbox input of HTML, but the style used in this case is a bit differently. For Ionic Framework to be tappable, then we have to create them within the "*<label>*" tag. Ionic Framework also supports the addition of colors to the checkboxes. Consider the example given below:

<ul class="list">

<li class="item item-checkbox">

<label class="checkbox">

<input type="checkbox">

</label>

Male

**
**

…

**
**

The above code will give you a checkbox similar to the one given below:

The checkbox will have a label named "*male.*"

Range

Colors can be added to the ranges, and then used in different elements. The code below shows how ranges can be created:

```
<div class="item range">
<i class="icon ion-volume-high"></i>
<input type="range" name="volume">
<i class="icon ion-volume-low"></i>
</div>
<div class="list">
<i class="icon ion-ios-sunny-outline"></i>
<div class="item range range-positive">
<input type="range" name="volume" min="0" max="99" value="35">
<i class="icon ion-ios-sunny"></i>
</div>
</div>
```

That is ranges can be implemented. An example of this is given below:

The above figure shows the range for the speaker of your mobile device. It can be used for

adjusting the volume of the speaker.

Select

This is much styled in Ionic Framework in such a way that it appears as the browser. This will be seen differently in the browser, in Android, and in iOS. Consider the code given below, which shows how the select can be implemented:

```
<div class="list">
<label class="item item-input item-select">
<div class="input-label">
Colors
</div>
<select>
<option>Blue</option>
<option selected>Green</option>
<option>Red</option>
</select>
</label>
</div>
```

On running the above code, you will be presented with a select component which has the three colors which we have specified. This is shown below:

That is how a select can be implemented in the Ionic Framework.

Chapter 6- Testing on Emulators, Browsers, and Mobile Devices

While developing your mobile app with Ionic Framework, testing the app on the browser is a bit easier compared to testing it on a mobile device. To debug the app when it is running, debugging it by use of the mobile device is a bit harder and not fun compared to when doing it with the browser. This is why it is recommended that you use the browser developer tools.

With the Ionic CLI, testing in the browser is easy. Begin by navigating to the root folder of your app. Once you are there, just execute the following command:

```
$ ionic serve
```

When the above command is executed, the HTTP server will be started on your system, which will be used to host your files in the "*www*" folder, and the default browser on your system will be launched for displaying this. The method is also in support of live-reload, so at the time the code for your app is changed, the app will be automatically refresh in your browser. What happens with the live-reload is that it will look for the changes which have been made to the files in the "*www*" folder and if any are found, the app will be refreshed in the browser but the "*lib*" folder is excluded.

Ionic Lab

Sometimes, you might need to view both the iOS and the Android versions of your app on the browser and next to each other. This can be done by use of the Ionic Lab which also supports the idea of the live-reload. Just execute the following command:

```
$ ionic serve --lab
```

We now need to look at how our apps look like at this moment. You can navigate to your views on both, and you will notice that there will be different transitions. In the case of the iOS, the text will be centered, while in Android, you will notice that the text will be displayed on the left. You will also notice the back buttons are not similar in both cases. That shows how interesting it is for one to use the Ionic Framework without having the need to implement some things not on your own. You will enjoy the native look of your app.

Adding Android and iOS platforms

We need to test our apps on Emulators and the real mobile device. However, we should begin by adding the Android and the iOS apps to the app. Just execute the following commands for this purpose:

```
$ ionic platform ios
$ ionic platform android
```

With the above commands, an Android folder and an iOS folder will be created, and these folders will be used to hold the actual xCode and the Android projects for your app. The projects will then be built into the packages which will be deployed into the app stores. Before one can test the apps, they should first be built. This can be done by use of the following command:

```
$ ionic build ios
$ ionic build android
```

Testing the App in the Emulator

It is highly recommended that one should test their app on an actual or real device since even it works on the emulator, this is not a guarantee that the app will work in the same way in the real device. The good thing aboutwith emulators is that they let you know the behavior of your app on different devices and on different operating systems.

For those who are developing the app for iOS and they need to launch it into the iOS simulator, run the following command:

```
$ ionic emulate ios
```

For Android users, begin by launching the emulator in Genymotion, and then run the following command:

```
$ ionic run android
```

In our case, you must have noticed that we have not used the "*ionic emulate android*" which is used for launching the default Android Emulator. With the Ionic CLI, the Genymotion emulator will be seen to be a real device, and that is why we have used the above command. For some of you, you might get the error given below:

Error executing "adb devices": ADB server didn't ACK
*** failed to start daemon ***

If that happens in your case, then move to the Genymotion and on the tab for “*ADB,*” select the option “*Use custom Android SDK tools.”* You should then select the path in which the installation of the SDK has been done.

That is the process of testing an app on the Emulator, whichcan be done in both Android and iOS.

Using a Mobile Device

As we said earlier on, testing our app on the real device will guarantee to us that the app will work in the environment. For this to be done, you should begin by connecting the mobile device to your computer by use of an USB cable.

For the app to be run on an iOS device, then one must possess an Apple iOS Developer account for which they will be charged $99 per year. The certificate for xCode should also be set. This account is the one to be used for deploying the app into the app store.

For Android users, then the process is easy for you, as you do not have to set up a Developer's account for you to be able to do this. The account will only be needed when one wants to publish their app to the Google play, and a registration fee of $25 will be incurred.

To test the app on the iOS, then run the following command:

```
$ ionic run ios
```

To test the app in Android, then run the following command:

```
$ ionic run android
```

Now you know how the testing can be done, both in the emulator or the simulator and the real mobile device.

Chapter 7- Development of the app

Now that you have learned the components which are available in Ionic Framework and how to test your app, it is good for you to learn how to develop a real app. We should begin by creating a folder structure which is easy to maintain in our "*www*" folder of the Ionic project. This is shown below:

index.html

/app

app.js

/login

login.controller.js

login.html

/dinners

ourdinners.controller.js

Our-dinners.html

/services

ourdinner.service.js

In my case, all of the Javascript code contained in the "*index.html*" file was moved to the "*app.js,*" and this is why we need to move the views into their own and separate "*.html*" files.

We need to be able to check on whether the login process was successful or not. Note that

the server will return a status code of 200, whether the login was successful or not. To perform the check, one header should be returned once the login process becomes successful.

Consider the code given below:

```
// ourdinner.service.js
angular.module('app').factory('DinnerService', ['$http', '$q', DinnerService])
function DinnerService ($http, $q) {
return {
login: function (username, password) {
var req = {
method: 'POST',
url: 'http://www.ourdinner.com/Account/LogOn',
headers: {
'Content-Type': 'application/x-www-form-urlencoded'
},
data: 'UserName=' + username + '&Password=' + password +
'&RememberMe=true'
};
return $http(req).then(function(resp){
var def = $q.defer();
// this header now available and after a successful log in process
if (resp.headers('x-xrds-location')) {
def.resolve();
}
```

```
else {
def.reject();
}
return def.promise;
});
},
getMyDinners: function () {
var pDinners = function (resp) {
var temp = document.implementation.createHTMLDocument();
temp.body.innerHTML = resp.data;
var it = temp.body.getElementsByClassName('upcomingdinners')[0].children;
var din = [];
for (var j = 0; j < it.length; j++) {
var it = items[j];
var dText = it.getElementsByTagName('strong')[0].innerText;
dText = dText.replace(/\r?\n|\r/g,'').replace(/\t+/, ' ');
var din = {
Name: it.getElementsByTagName('a')[0].innerText,
Date: moment(dText, 'YYYY-MMM-DDhh:mm A').toDate(),
Location: it.innerText.split('at')[1]
};
din.push(din);
}
return din;
}
return $http.get('http://www.ourddinner.com/Dinners/My')
.then(function(resp){
```

```
return parseDinners(resp);

});

}

}

}
```

The Controller for Login

This should be the controller for handling the login view. If the login process in our system runs successfully, then we should be taken to the state "*our-dinners.*" However, if the login process runs unsuccessfully, then we should show up a popup by use of the "*$ionicPopup*." The code for this is given below:

```
// login.controller.js

angular.module('app').controller('LoginController', ['$state', '$ionicPopup', 'DinnerService', LoginController]);

function LoginController($state, $ionicPopup, dinnerService) {

var dm = this;

dm.doLogin = function () {

var ifSuccess = function () {

$state.go('our-dinners');

};

var ifError = function () {

$ionicPopup.alert({

title: 'The Login did not succeed :(',

template: 'Please try again!!.'

});

};

dinnerService.login(dm.username, dm.password)

.then(ifSuccess, ifError);

}
```

}

With the above code, the login process will be well cared for.

We also need a controller for the view. This should be implemented as shown below:

// ourdinners.controller.js

angular.module(‘app’).controller(‘DinnersController’, [‘DinnerService’, DinnersController]);

function DinnersController(dinService) {

var dm = this;

dinService.getMyDinners().then(

function (din) {

dm.din = din;

});

return dm;

}

The next step should be to take the code for our views out of the “*index.htm”* and then put it in a separate file of its own. The data binding expressions should then be added so as to communicate with the controllers which are corresponding. The script tags need to be completely removed from the view templates. Consider the code given below for doing all of this:

<!— login.html —>

```
<ion-view title="Login" ng-controller="LoginController as dm">
<ion-content padding="true" scroll="false" class="has-header">
<form>
<ion-list>
<label class="item item-input">
<span class="input-label">Your Username</span>
<input type="text" placeholder="" ng-model="dm.username">
</label>
<label class="item item-input">
<span class="input-label">Your Password</span>
<input type="password" placeholder="" ng-model="dm.password">
</label>
</ion-list>
<div class="spacer" style="width: 350px; height: 20px;"></div>
<a ng-click="dm.doLogin()" class="button button-light button-block">Log in</a>
</form>
</ion-content>
</ion-view>
```

That is the code for the login view. The next code should be as follows, and this should be for the view for Dinners:

```
<!— our-dinners.html —>
<ion-view title="Our Dinners" ng-controller="DinnersController as dm">
<ion-content padding="true" class="has-header">
```

```
<div class="list card" ng-repeat="din in dm.din">
<div class="item item-divider">{{ din.Name }}</div>
<div class="item item-body">
<div>
<div>
<i class="icon ion-calendar"></i>{{ din.Date }}</div>
<div>
<i class="icon ion-location"></i>{{ din.Location }}</div>
</div>
</div>
</div>
</ion-content>
</ion-view>
```

The routing code in the "*app.js*" should also be changed so that it points to the location of the templates. This can be done as shown in the next code:

```
// app.js
…
$stateProvider
.state('login', {
url: '/login',
templateUrl: 'app/login/login.html'
})
.state('our-dinners', {
```

url: ‘/our-dinners’,

templateUrl: ‘app/din/our-dinners.html’

});

// this will be used as a fallback if none of the above states is matched.

$urlRouterProvider.otherwise(‘/login’);

…

After the above steps, the app will be ready for testing.

This can be done as follows:

We need to start our testing in the desktop browser. Since we are using the same origin policy, then we have to disable the security in the browser. This means that after loading the app with the “*ionic serve,”* a separate browser with the flag *“—disable-security*” has to be opened to the URL in the browser.

For those who are doing it in OSX, then execute the following command:

```
$ open -a Google\ Chrome --args --disable-web-security
```

For Windows users, use the following command:

With Ionic, a proxy can be configured which can be used for bypassing the URL so that it takes you to the external URL. In my case, I also need to send a cookie to my server so that once I am logged in, it will know that the process was successful.

Chapter 8- The Ionic CLI

With the Ionic CLI, users usually find it easy for them to start, build, run, and emulate their Ionic apps. Mobile development services are expected to be supported by this tool in the future. If you need to get help on how to use this tool, just run the command "*ionic –help"* or "*ionic help."* The input for the CLI is received from the shell, and then displayed on the same shell.

Installation of the Ionic CLI

The easiest way how one can install this tool is by use of the "*npm"* tool. This is shown below:

```
$ npm install -g ionic
```

For the users of OSX and Linux, then you might be forced to use the "*sudo"* command in front of the above command.

The next step should involve updating of the Ionic Library files. These files can be found in the directory "*www/lib/ionic.*" In case your project is using the bower, then the "bower update ionic" will be executed automatically. Otherwise, the command will update the local static files from the CDN of Ionic. The following command can be used for this purpose:

```
$ ionic lib update
```

Note that with the above command, the Ionic dependencies will not be updated. However, if you need to update these together with the Ionic, just execute the command "*bower update.*"

If you need to start your Ionic app, then just run the following command:

```
$ ionic start myapp [template]
```

For the starter templates, they can come from template which is named, a Codepen, a GitHub repo, or just a local repository. With the starter template, once you have the cordova template, then it becomes the directory *www.*

If you need to start the local development server for the purpose of app development and testing, then use the command "*ionic serve.*" This is very useful when it comes to testing on the desktop browser, as well as testing within a device browser which has been connected to the network. The command will also launch the live-reload feature which can be used to monitor changes live as they are done to the files. Once the file has been saved, then the browser will automatically be reloaded. The command is given below:

```
$ ionic serve [options]
```

LiveReload

The default setting is that this feature will match for the changes which are made in the directory "*www/*" but the directory "*www/lib*" is excluded. If you need to specify this, then a specification needs to be done in the "*watchPatterns*" property of the Ionic project. This can be found in the root of the project, and you will be in a position to watch for the changes as they are being made to the files. This is shown in the code given below:

{

"name": "ourApp",

"app_id": "",

"watchPatterns": [

"www/js/*",

"!www/css//*"**

]

}

Consider the command given below:

```
$ ionic setup sass
```

With the above command, the "*watchPatterns*" property will be added to the "*ionic.project*" with its default values.

If you need to add some proxies to your HTTP server, then use the "*serve*" command. The proxies will be of great importance for those who are developing within the browser, and if they need to make their calls to an external APIs. Consider the code given below:

{

"name": "applicationname",

"email": "",

"app_id": "",

"proxies": [

{

"path": "/v1",

"proxyUrl": "https://api.facebook.com/v1"

}

]

}

The above code makes it possible for you to make requests to the local server which is located at "*http://localhost:8100/v1*." This will also make it possible for you to proxy the "*https://api.facebook.com/v1*." Consider the example given below:

angular.module('starter.controllers', [])

.constant('FacebookApiUrl', '')

// .contant('facebookApiUrl','https://api.facebook.com')

```
//making the real URL for the production environment.

.controller('FeedCtrl', function($scope, $http, FacebookApiUrl) {

$scope.feed = null;

$http.get(InstagramApiUrl + '/v1/media/search?
client_id=1&lat=48&lng=2.184381').then(function(d) {

console.log('data ' , d)

$scope.feed = d;

})

})
```

Chapter 9- Routing

We need to add two new routes to the application, which are the states. One should be for loading the session view, and the other one should be for loading the session details view.

The first step should be for defining the app.sessions route, which should be done by following the steps given below:

1. In your "*conference/www/js,*" open the "*app.js.*"
2. Delete the state "*app.playlists.*"
3. The deleted state should then be replaced with the state "*app.sessions,*" which is defined below:

.state('app.sessions', {

url: "/sessions",

views: {

'menuContent': {

templateUrl: "templates/sessions.html",

controller: 'SessionsController'

}

}

})

The next step should be defining the "*app.sessions"* route. This can be done by following the steps given below:

1. Delete the state “*app.single.*”
2. This should then be replaced with the app.session state. This is given below:

.state(‘app.session’, {

url: “/sessions/:sessionId”,

views: {

‘menuContent’: {

templateUrl: “templates/session.html”,

controller: ‘SessionController’

}

}

});

The next step should involve the modification of the default route, which can be done as follows:

```
$urlRouterProvider.otherwise('/app/sessions');
```

Once you execute the above command, the fallback route will be modified to the list of sessions, which is the last line contained in the app.js file.

The side menu should then be modified, which can be done as follows:

1. Navigate to the directory “conference/www/templates,” and then open the file “*menu.html.”*

2. The Playlists menu item should then be modified as follows:

   <ion-item menu-close href=”#/app/sessions”>

   Sessions

   </ion-item>

   Both the item label and the href items should be modified.

Once you are done with the above steps, the app can be tested. This can be done by following the steps given below:

1. Begin by ensuring that the local web server (Ionic server) is running. If it is running and the app page on the browser has been closed, the app can be reloaded by running the following URL on the browser: http://localhost:8100.

   However, if the server is not running, launch the command prompt, and then use the “*cd”* command so as to navigate to the “*ionic”* directory. The following command can then be executed:

   ```
   ionic serve
   ```

2. In your conference application, launch the side menu and then select the

“*Sessions.”* Once you have selected the session from the details, the details of the session will be presented to you.

Chapter 10- Integrating your App with Facebook

You should know how to integrate your application with Facebook. Users will be allowed to log in by use of their Facebook account, view their profile, and they will be in a position to share their favorite sessions on their feed.

In this chapter, the integration will be done by use of "*OpenFB.*" This is just a micro-library with the ability of integrating your application written in Javascript with Facebook. Both the Cordova/PhoneGap and browser based apps are supported. The application supports no dependency. If you are running it in Cordova, then you will not need to have the Facebook plugin. The Facebook SDK will also not be needed.

The Facebook application can be created by following the steps given below:

1. Login to your Facebook account.

2. Navigate to the "*https://developers.facebook.com/apps*," and then click on the link labeled "*Add New App.*"

3. The selected platform should be the *www.*

4. Provide a name for your app, and the name must be unique. When you are done, click on the link labeled "*Create New Facebook App ID.*"

5. Specify the necessary category, and then click on "*Create App ID.*"

6. From your Menu, click on "*My Apps,*" and then select the name of the app that you have just created.

7. From the navigation on the left window, click on "*Settings.*"

8. Click on the tab written "*Advanced Tab.*"

9. In the section for "*OAuth Settings*" and the field for "*Valid OAuth redirect URIs*",just provide the following URLs:

    - http://localhost:8100/oauthcallback.html- this will be used for accessing the system by use of the Ionic serve.
    - https://www.facebook.com/connect/login_success.html- this will be used for accessing the app from the Apache Cordova.

10. *Once you are done, just click on the button labeled"Save changes."*

Initialization of the OpenFB

This can be done by following the following steps:

1. The OpenFb files should be added to the application. This can be done as follows:

    - ngopenfb.js and openFB.js should be copied from the "*ionic-tutorial/resources*" to the "*conference/www/js.*"
    - "*logoutcallback.html*" and "*oauthcallback.html*" should be copied from the "ionic-tutorial/resources" to the "*conference/www/js.*"
    -

    Script tags should then be added to "*conference/www/index.html."* The following are the script tags:

<script src="js/openfb.js"></script>

<script src="js/ngopenfb.js"></script>

2.

The "conference/www/js/app.js" should now be opened, and then the "*ngOpen()"nFB"* added as the dependency to the starter. This can be done as shown below:

```
angular.module('starter', ['ionic', 'starter.controllers', 'ngOpenFB'])
```

3.

The ngFB should then be injected into the declaration of the "*run()*" function as shown below:

```
.run(function ($ionicPlatform, ngFB) {
```

4.

The ngFB should then be initialized in the first line of the run() function. This can be done by replacing the "YOUR_FB_APP_ID" with the following:

```
ngFB.init({appId: 'YOUR_FB_APP_ID'});
```

Adding Facebook Login

This can be done as follows:

1.

Navigate to the directory "*conference/www/templates,"* and then open the file "*login.html."* After the "*Login"* button, add another button and give it the name "*Login via Facebook."* This is shown below:

<label class="item">

<button class="button button-block button-positive" ng-click="fbLogin()">

Login via Facebook

</button>

</label>

2.

The file "*conference/www/js/controllers.js*" should then be opened, and the ngOpenFB added as a starter to the module "*starter.controllers."* This is shown below:

```
angular.module('starter.controllers', ['starter.services', 'ngOpenFB'])
```

3.

The ngFB should then be injected into the controller “*Appctrl”* as shown below:

```
.controller('AppCtrl', function ($scope, $ionicModal, $timeout, ngFB) {
```

4.

The function “*fbLogin”* should then be added to the controller “*AppCtrl”* as shown below:

```
$scope.fbLogin = function () {
ngFB.login({scope: ‘email,read_stream,publish_actions’}).then(
function (resp) {
if (resp.status === ‘connected’) {
console.log(“Logged in successfully”);
$scope.closeLogin();
} else {
alert(‘Facebook login was not successful’);
}
});
};
```

5.

Once done, the application can be tested. This can be done as follows:

- Ensure that the Ionic serve is up and running.
- On the side menu of the application, select “*Login.”*
- Click on the button written “*Login via Facebook.”*
- Provide the credentials for your Facebook login, and then authorize the application to login.
- Open the console of your browser, and you will observe the successful login message on it.

If the above steps run successfully, then know that you are set.

Displaying the profile of the User

This can be done by following the steps given below:

1.

Begin by creating a template for the user profile view. In the directory "*conference/www/templates,*" create a new file and then give it the name "myProfile.html." Add the following code to the file:

<ion-view view-title="MyProfile">

<ion-content class="has-header">

** **

** **

<h2>{{user.name}}</h2>

** {{user.town}} **

** **

** **

****

** **

** **

</ion-content>

</ion-view>

2.

A controller for the user profile view should then be created. This can be done by opening the controller “controllers.js,” and then adding the following code to it:

```
.controller(‘ProfileCtrl’, function ($scope, ngFB) {
ngFB.api({
path: ‘/mine’,
params: {fields: ‘id,name’}
}).then(
function (usr) {
$scope.usr = usr;
},
function (err) {
alert(‘Facebook err: ‘ + err.error_description);
});
});
```

3.

A route for the user profile view should then be created. Just open “*app.js”* and then add the following code to it:

```
.state(‘app.myprofile’, {
url: “/myprofile”,
views: {
```

‘menuContent’: {

templateUrl: “templates/myprofile.html”,

controller: “ProfileCtrl”

}

}

})

4.

The following menu item should then be added to “*www/templates/menu.html”:*

<ion-item menu-close href=”#/app/myprofile”>

MyProfile

</ion-item>

5. Once you are done with the above steps, the app can be tested. This can be done by following the steps given below:

    - Ensure that the Ionic serve is up and running.
    - On the side menu of the application, select “*Login.”*
    - Login with Facebook.
    -

Provide the credentials for your Facebook login, and then authorize the application to login.

-

From the side menu, just click on “*MyProfile.*”

Publishing to the Feed

This can be done as follows:

1.

Open the file "*controller.js,"* and then inject the ngFB into the definition of SessionCtrl as follows:

```
.controller('SessionCtrl', function ($scope, $stateParams, Session, ngFB) {
```

2.

The "*share()"* function should then be added as follows:

$scope.share = function (ev) {

ngFB.api({

method: 'POST',

path: '/mine/feed',

params: {

message: "I will be at the: '" + $scope.session.title + "' by " +

$scope.session.speaker

}

}).then(

function () {

alert(“The sharing on Facebook was successful”);

},

function () {

alert(‘The sharing on Facebook did not succeed due to an error.’);

});

};

3.

Navigate to the directory for templates, and then add the handler “*ng-click”* to the share button. The share function can then be invoked as follows:

****

<i class=“icon ion-share”></i>

Click to Share

****

4.

Once done, the application can be tested. This can be done as follows:

- Ensure that the Ionic serve is up and running.
- On the side menu of the application, select “*Login.”*

Login with Facebook.

-

On the side menu, just click on “*sessions,*” and then select a particular session from the list.

-

Tap on the “*Share*” button.

-

Check on the feed on Facebook.

One can now test for the Facebook integration on a device. This can be done by following the steps given below:

1.

Begin by adding the “InAppBrowser” which is used by OpenFB when it is running in Cordova. You can then use the command line to navigate to the following directory:

ionic-tutorial/conference.

The following command can then be executed*:*

```
cordova plugins add org.apache.cordova.inappbrowser
```

2.

You can then build your app for the platform in which you are building it for. The following commands can be used:

```
ionic build ios
```

```
ionic build android
```

3.

The application can then be run on an Android or iOS emulator or device.

Conclusion

It can be concluded that the Ionic Framework is a very useful framework. It is used for development of hybrid mobile applications for our devices. The applications developed by use of this framework are hybrid, and this is why developers like to use it for development of mobile apps. With Ionic, one can develop apps for their mobile devices without having to learn native programming languages such as Swift, Java, and Objective-C. The framework is an open-source one, meaning that you can download and use it for free on your system. The hybrid apps, which can be developed by use of the Ionic frameworks are of a greater advantage compared to the native ones since with the former, the speed of development as well as the performance of the app is greatly improved.

Before beginning to develop your apps with Ionic, you should begin by setting up your environment, by downloading and installing it on your system. Note we said that the SDK is open-source. It is after setting up the environment that you can get into the programming. Note that when it comes to testing of the apps, you can choose to use the emulator or the simulator or use the actual device. However, there are advantages associated with the latter case. That is why it is recommended that you use the real device for the purpose of testing.