Beyond PhoneGap – Ionic

In this chapter, we will go a bit further by learning how to apply a hybrid mobile UI that will work across different platforms and screen sizes. Ionic framework seems like a great candidate for use along with PhoneGap. We will cover the following topics:

- Introduction to the Ionic framework
- App development process
- Creating a to do application

Introduction to the Ionic framework

The PhoneGap framework does not come with any predefined frameworks for code structure or a frontend user interface. In this area, it is completely agnostic. As we described it in the previous chapter, there are many solutions to select from and many can be mixed together. One of the recent frameworks was the Ionic framework that gained a lot of popularity and traction since it was launched in 2013.

What is Ionic?

Ionic is a completely free and open source framework for hybrid mobile application development. It is built with previously available components such as PhoneGap and AngularJS. Ionic is a fully functional SDK that comes together with the **command-line interface** (**CLI**), tools, and services to develop hybrid applications with the same technologies that work on top of PhoneGap. It was created by a team of developers from Drifty Co. in 2013.

Each Ionic app is essentially a website running in a browser shell that has access to a native platform layer through PhoneGap. It has all the benefits that PhoneGap offers in addition to creating a fully mobile experience through the available UI components. Ionic needs to have a native wrapper to work with such as PhoneGap. It also works perfectly with Apache Cordova, which is the default selection for its CLI. With the use of Ionic, it is really easy to create production quality applications on top of JavaScript, HTML5, and CSS3.

Ionic has just come in at the right time to reap the rapid progress of mobile platforms. Until recently, they had limited processing power and rendering on mobile screens was slow. Solutions for native SDKs such as iOS or Android were much more advanced and detailed to write mobile native apps as compared to the web apps that were mostly controlled by jQuery. The best example that mobile web apps weren't ready back then was in 2011 when Facebook tried to publish their app with web technologies.

Before Ionic came to light, it was hard to have a native feeling UI with any frontend library for hybrid apps. AngularJS changed the paradigm by introducing a different way to build web applications in general. The team at Drifty Co. had seen the benefits of this and used it to build a framework that worked specifically for mobile screens with its UI components that work on mobile devices.

App development process

Ionic is a collection of AngularJS UI Router, Angular directives, Angular services, JS utilities, and mobile focused CSS styles. These are bundled together as `ionic.bundle.js` and `ionic.css`. This chapter doesn't directly require the knowledge of AngularJS, but it is advisable to go over basic AngularJS concepts before starting with this chapter. The best way to learn more about this is to go to the official page that can be found at `https://angular.io/docs/js/latest/quickstart.html`. It will explain the basic concepts that might help you to understand how things work in Ionic. Ionic can be used without AngularJS but as a lot of the JavaScript core functionality works through it, it is better to get to know it if you plan to use it.

Installing Iconic

As this is a quick introduction to Ionic, we will use the CLI to go over the details, which means we will need to run Ionic with Cordova instead of PhoneGap. Ionic CLI does not support PhoneGap out of the box, so it requires using the template to run for a new PhoneGap project. Many of the commands running from the Ionic CLI are forwarders to the Cordova CLI.

Let's start with the minimum requirements to build a hybrid mobile app with the current release. It should support any iOS version from 6.0 and Android version 4.1, though there are noticeable lags on some Android versions for some devices; so before deploying the app, it is advisable to install it on many different Android devices to test the behavior.

The Ionic apps can be developed on any operating system that you prefer as long it is possible to install Node.js. We will use Mac OS in this chapter, but all the commands are tested and can work on any major platform. You should follow the installation instructions from the first chapter before proceeding with this one. Make sure you have a working Git and Node.js before running the following commands.

There is a single command to install if you have correctly followed the installation process in *Chapter 1*, *Get up and Running with PhoneGap*. If you are running a Windows installation, please omit the `sudo` keyword from the installation command, as follows:

```
sudo npm install -g cordova ionic
```

After the installation process is done, you should run the `ionic` command to verify the installation:

```
ionic
```

You should get the similar output as I did, only the Ionic version might differ in the future. The following is the output that we should get:

```
ruby-2.2.1 ~/Projects (master) $ ionic
  _                 _
 (_)               (_)
  _  ___  _ __   _  ___
 | |/ _ \| '_ \| |/ __|
 | | (_) | | | | | (__
 |_|\___/|_| |_|_|\___|  CLI v1.6.4

Usage: ionic task args

=======================

Available tasks: (use --help or -h for more info)
```

Now, you are ready to use it.

Introducing the command-line interface

Let's get to know more about the command-line interface. First, we will need to run the following command to start a new project:

```
ionic start projectName templateName
```

The `projectName` is the name of the actual project and `templateName` is the name of the template for the initial project. Currently, there are three project templates available.

A blank project template

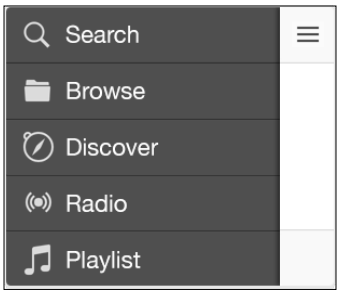
A blank project doesn't contain any additional code and it is completely empty besides preparing the project to start with. To use this template as a starting point, you use `blank` for `templateName`. You will get the following output:

A tab based project template

A tab based project has already prepared the initial structure to support and run as a tab based application. So every view is run in the tab selections. To use this template as a starting point, you use `tabs` for `templateName`. You will see the following screenshot:

A side menu project template

A side menu project template has been already prepared in order to have the initial structure to support the side menu on the left-hand side to select different actions. To select different actions, you need to open the side menu and select from it. To use this template as a starting point, you use `sidemenu` for `templateName`. It displays the following:



Creating your first application

Now that we have installed everything, let's create our first application and run it in an Android simulator in order to verify that everything is working, and we will also learn about the commands to run it.

Let's create our first `HelloWorld` mobile application with Ionic that has tab navigation built in. To do so, perform the following steps:

1. Run the following command:

   ```
   ionic start HelloWord tabs
   ```

   It will create a new project with the name, `HelloWorld`, that uses the `tabs` project template. Now we will build and run it to show the whole process of running it.

2. Let's enter in the project folder itself:

   ```
   cd HelloWorld
   ```

3. Now, add an Android platform to be able to run it on an Android device or emulator:

   ```
   ionic platform add android
   ```

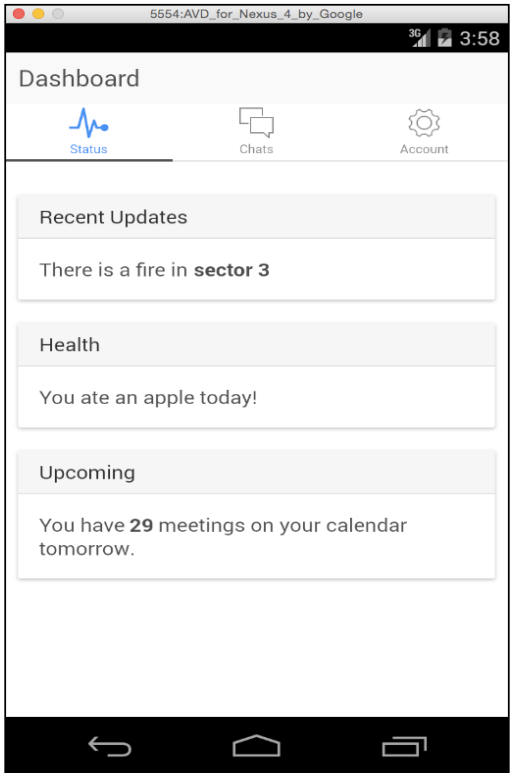
4. Then we need to build the project with the following command:

   ```
   ionic build android
   ```

5. After the project has been built, we can run it in the emulator. Make sure that you have followed the installation in the first chapter so that you already have an Android emulator set up; otherwise it will return an error:

```
ionic emulate android
```

6. Running the preceding command should open the emulator and run the project. You should get a screen similar to the following screenshot:



As you can see, most of the commands are quite similar to the running of the plain PhoneGap command-line interface. This is because most of the build process is delegated to the wrapper.

Wrapper – ngCordova

The `ngCordova` library is one of the most useful libraries to be used with Ionic. The `ngCordova` library is a wrapper for the PhoneGap plugins. It makes the PhoneGap plugins work as an Angular service so that you can use them for dependency injection. It simplifies the process of handling the different native plugins so that they behave like any other Angular service library.

It provides a variety of functionalities that are available in the PhoneGap applications such as taking photos, handling push notifications, geolocation services, and progress indicators. Currently, it has over 70 common features available. The best part is that you do not need to write any event listeners to know when PhoneGap is ready and you can use the plugins. Once you inject them in the Angular code, it will be prepared and ready to work. You can read more about this useful library at `http://ngcordova.com/`.

Creating a to do application

One of the first applications that we usually learn the basics from is the to do application. It is a simple application but it shows the basic concepts of a new technology. We will build a simple single page to do application that can add new items to the scrollable list.

First, let's create a new project by running the following command:

```
ionic start todo blank
```

Next we will go in the project root folder:

```
cd todo/
```

As this is going to be a quick introduction, we will write all the JavaScript code in the same file. For JavaScript, we will use `www/js/app.js`. There should already be some code that we will leave alone as it initializes some of the default behavior. We will just change, as follows:

```
var app = angular.module('todo', ['ionic'])
app.run(function($ionicPlatform) {
  $ionicPlatform.ready(function() {
    // Hide the accessory bar by default (remove this to show the
accessory bar above the keyboard
    // for form inputs)
    if(window.cordova && window.cordova.plugins.Keyboard) {
      cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
    }
    if(window.StatusBar) {
      StatusBar.styleDefault();
    }
  });
});
```

A to do Model

First, we will create a new angular service module to behave as a model part for our application. It will contain a list of all the to do items and the ability to add or remove an item from the list. It should be added after the previous code, as follows:

```
app.service('Todo', function() {
  var todos = [
    {
      title: 'Buy Milk',
      done: false
    },
    {
      title: 'Make an Appointment',
      done: false
    }
  ];

  this.list = function() {
    return todo;
  };

  this.add = function(todo) {
      todos.splice(0, 0, todo);
  };

  this.remove = function(index) {
    todos.splice(index, 1);
  };
});
```

We created a new Angular service that has two default to do items on initialization and two functions: to add and remove the items from the list.

A to do Controller

We need to create a controller that will interact between our to do model and the view layer. We will inject our to do service into it. The controller loads all the available to do items and has an `addItem` function to add a new to do item and `removeItem` to remove a to do item, as shown in the following code:

```
app.controller('TodoController', ['$scope', 'Todo', function($scope,
Todo) {
  $scope.todos = Todo.list();
```

```
  $scope.addItem = function(todo) {
    todo.done = false;
    Todo.add(angular.copy(todo));
    todo.title = '';
  };

  $scope.removeItem = function(index) {
    Todo.remove(index);
  };
}]);
```

In the specified scope, as there can be only one object with the same key name, we will reuse it every time and use the `angular.copy` function to clone the object that will be actually inserted in the to do list.

The presentation page

We will change the default `index.html` web page to contain the code to present the to do page, as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-
      scale=1, user-scalable=no, width=device-width">
    <title></title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="todo">
    <ion-pane>
      <ion-header-bar class="bar-stable">
        <h1 class="title">Todo list</h1>
      </ion-header-bar>
      <ion-content ng-controller="TodoController">
      <!-- {{todos}} -->
        <div class="list">
          <label class="item item-input">
            <input type="text" placeholder="New Todo" ng-
              model="todo.title">
```

```
            </label>
            <button class="button button-block button-positive" ng-
              click="addItem(todo)">Add</button>
            <ion-checkbox ng-repeat="item in todos" ng-
              model="item.done" ng-checked="item.done" ng-
                change="removeItem($index)">
              {{item.title}}
            </ion-checkbox>
          </div>
        </ion-content>
      </ion-pane>
    </body>
  </html>
```

In this example, we used some of the Ionic directives in an HTML5 code. This is a part of the powerful features coming from Angular. You can create new tags that will render this functionality.

In this example app, you can add new to do items, and by clicking on the checkbox, they can be removed. This example shows you how you can build a powerful application that works across various platforms with just a few lines.

Let's describe some of the following directives that were used in this page:

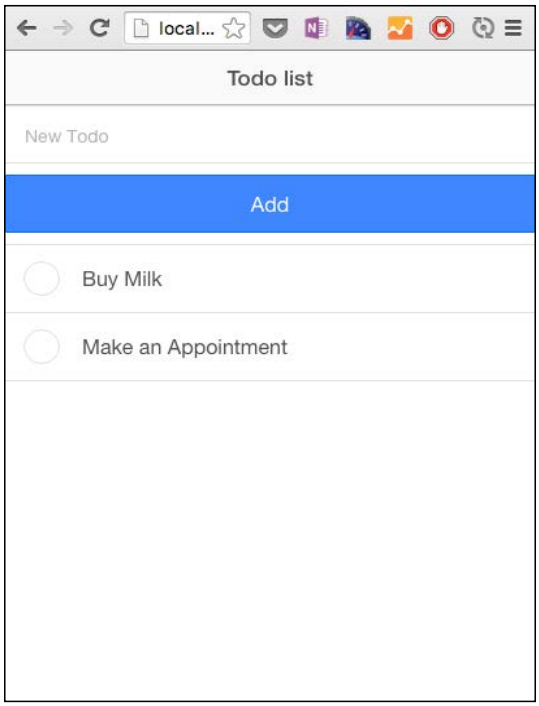
- `ion-pane`: This is a simple container that fits content with no side effects. It also adds the `pane` class to the element.
- `ion-header-bar`: This adds a fixed header bar at the top where you can add the title or two action buttons on each side.
- `ion-content`: This provides a content area that can use the scroll functionality of Ionic. It is the main content area to present information.
- `ion-checkbox`: This creates a list item that has the clickable checkbox with the provided text information.

Preview

There are two ways to test the application that we just created: we can test it with the same commands that we used for the `HelloWorld` application and run it in the emulator or we can use the following command:

```
ionic serve
```

It will create a local web server and present the mobile application in your desktop browser, which behaves like a mobile application with the UI and interaction. It is a great way to test and build the features as it automatically reloads in order to run the changed code in real time. The following is the screenshot of the application:



Summary

In this chapter, you learned the basics about the Ionic framework and how it can be used on top of the wrapper. Then we created a basic to do application so as to showcase the basic concepts of the framework, which can be used to create a hybrid mobile application for multiple platforms that have the unified mobile UI with the shared logic behind them.