

Rahat Khanna, Sani Yusuf,  
Hoc Phan

# Ionic : Hybrid Mobile App Development

## Learning Path

Create cutting-edge, hybrid mobile applications using the Ionic framework



Packt

# Ionic : Hybrid Mobile App Development

Create cutting-edge, hybrid mobile applications using the Ionic framework

A course in three modules

**Packt**

BIRMINGHAM - MUMBAI

# Ionic : Hybrid Mobile App Development

Copyright © 2017 Packt Publishing

All rights reserved. No part of this course may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this course to ensure the accuracy of the information presented. However, the information contained in this course is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this course.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this course by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Published on: June 2017

Production reference: 1140617

Published by Packt Publishing Ltd.

Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78829-311-2

[www.packtpub.com](http://www.packtpub.com)

# Credits

**Authors**

Rahat Khanna  
Sani Yusuf  
Hoc Phan

**Reviewers**

Nikola Brežnjak  
Mike Hartington  
Luca Mezzalira  
Ted Morin  
Chady Kassouf  
Siva Prakash  
Syed Iqrar Raza Zaidi

**Content Development Editor**

Arun Nadar

**Graphics**

Jason Monteiro

**Production Coordinator**

Arvindkumar Gupta



# Preface

Ionic has evolved as the most popular choice for Hybrid Mobile App development as it tends to match the native experience and provides robust components/tools to build apps. The Ionic Complete Developers course takes you on an end-to-end journey, empowering you to build real-time, scalable, and interactive mobile applications with the Ionic framework.

Starting with an introduction to the Ionic framework to get you up and running, you will gradually move on to setting up the environment, and work through the multiple options and features available in Ionic to build amazing hybrid mobile apps. You will learn how to use Cordova plugins to include native functionality in your hybrid apps. You will work through three complete projects and build a basic to-do list app, a London tourist app, and a complete social media app. All three projects have been designed to help you learn Ionic at its very best. From setting up your project to developing on both the server side and front end, and best practices for testing and debugging your projects, you'll quickly be able to deliver high-performance mobile apps that look awesome.

You will then hone your skills with recipes for cross-platform development. Integrating Ionic with Cordova will bring you native device features, and you will learn about the best modules from its ecosystem. Creating components and customizing the theme will allow you to extend Ionic. You'll see how to build your app to deploy to all platforms to make you a confident start-to-finish mobile developer.

## What this learning path covers

*Module 1, Getting Started with Ionic,* This module equips any web developer with the practical knowledge required to use modern web technologies in building amazing Hybrid Mobile Apps using Ionic. This fast-paced, practical module that explains all the important concepts of AngularJS and Cordova framework required to develop Ionic apps. Then, it gives you a brief introduction to Hybrid Mobile Applications. It will guide you through setting up the development environment for different mobile platforms and through the multiple options and features available in Ionic, so you can use them in your mobile apps. Features, such as the side menu, tabs, touch interactions, and native features, such as bar code, camera, and geolocations, are all covered. Finally, we'll show you how to use Cordova plugins and use Ionic cloud services to empower your mobile apps.

*Module 2, Ionic by Example,* This module is a step-by-step guide that covers the very basics of Ionic aiming to equip the reader with all the necessary knowledge needed to understand and create Ionic apps. You will start off by learning a bit about the history of Ionic, and then slowly learn to get it set up and work with its great features. You will learn to work with Ionic and create four different Ionic apps, with each app teaching you different important features of Ionic. You will also learn to connect your app to a database using Firebase. This module will also provide you with links to some great resources to further your quest for more advanced Ionic knowledge.

*Module 3, Ionic Cookbook,* takes you through the process of developing a cross-platform mobile app using just HTML5 and the JavaScript-based Ionic. You will start with an introduction to the CLI and then move on to building and running an app. You will explore common features of real-world mobile apps such as authenticating a user, and getting and saving data using either Firebase or Local Storage. Next, the book covers how Ionic integrates with Cordova to support native device features using ngCordova, and you will discover how to take advantage of existing modules around its ecosystem. You will also delve into advanced topics, including how to extend Ionic to create new components. Finally, the book will walk you through customizing the Ionic theme and building the app so that it can be deployed to all platforms.

## What you need for this learning path

### Module 1:

For this module, you require a system with Windows, Mac, or Linux OS. You need to install NodeJS and NPM to manage dependencies for different projects. This module will guide you through the setup for mobile app development platforms for iOS and Android. Test devices having Android and iOS would be required to test the mobile apps.

### Module 2:

Firstly, you will need a Windows, Linux, or Mac computer to follow the code samples in this module. A beyond basic or intermediate knowledge of JavaScript and HTML5 is certainly essential to understand concepts discussed in this module. A basic understanding of Cordova is expected at the very least. You are expected to also have an idea of how to issue commands in a terminal window. You should also have access to a working Internet connection, and a Google account is necessary for Chapter 9, Connecting to Firebase.

### Module 3:

- A Mac computer with Mac OS X Yosemite and root privilege
- Or a PC with Windows 7 or later with Administrator privileges
- iPhone 5 or later
- An Android device with Android 5.x or later (optional)
- A Windows phone device (optional)

## Who this learning path is for

If you are a web developer with basic knowledge of working with modern web technologies, this comprehensive course is best-suited to empower to build hybrid mobile app development using the Ionic framework.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course – what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a course, see our author guide at [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for this course from your account at <http://www.packtpub.com>. If you purchased this course elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Ziipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at <https://github.com/PacktPublishing/Ionic-Hybrid-Mobile-App-Development>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the course in the search field. The required information will appear under the **Errata** section.

## Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

## Questions

If you have a problem with any aspect of this course, you can contact us at [questions@packtpub.com](mailto:questions@packtpub.com), and we will do our best to address the problem.



## **Module 1: Getting Started with Ionic**

---

<b>Chapter 1: All About Hybrid Mobile Apps and Ionic Framework</b>	<b>3</b>
Introducing a Hybrid Mobile Application	4
Using web technologies to develop for mobile	8
What is AngularJS?	8
Why use Apache Cordova?	13
Introducing Ionic Framework	14
Summary	14
<b>Chapter 2: Setting up the Environment the Right Way</b>	<b>15</b>
Setting up Native Mobile development environments	16
Ionic development environment	20
Basic Ionic CLI commands	22
Building a dummy app	24
Popular issues faced and solutions	26
Alternative to installation fuss – Ionic Playground	27
Using Ionic Framework with different Code Editors	27
Summary	28
<b>Chapter 3: Start Building Your First Ionic App</b>	<b>29</b>
Starting a new project	30
Multiple ways to start a project	30
The anatomy of Ionic Project	35
The Ionic starter template	40
E-commerce sample app – BookStore	44
Summary	45

*Table of Contents*

---

<b>Chapter 4: Navigation and Routing in an Ionic App</b>	<b>47</b>
Introduction to Angular UI Router	48
Ionic header and footer	55
Ionic Tabs	57
Ionic side menu	59
Navigation and back menus	61
Navigation and layout to be used in BookStore	61
Summary	62
<b>Chapter 5: Accessorizing Your App with Ionic Components</b>	<b>63</b>
Ionic CSS components	64
Ionic JS components	73
Miscellaneous components	82
Summary	83
<b>Chapter 6: Integrating App with Backend Services</b>	<b>85</b>
\$http services	86
Ionic services vs factories	88
\$resource and REST API	92
Demystifying mBaaS	93
Integrating with Parse	94
Integrating to Firebase	96
Summary	97
<b>Chapter 7: Testing App on Real Devices</b>	<b>99</b>
Testing on browser emulators	100
Ionic view app	102
Making debug build	104
Remote debugging	106
Testing using Ngrok	109
Summary	110
<b>Chapter 8: Working with Cordova Plugins – ngCordova</b>	<b>111</b>
Introduction to Cordova plugins	112
Integrating ngCordova to Ionic App	113
Important plugins	114
Network	122
Custom Cordova plugin development	127
Summary	127
<b>Chapter 9: Future of Ionic</b>	<b>129</b>
Ionic cloud services	130
Ionic v2	136
Summary	138

---

## **Module 2: Ionic Framework By Example**

---

<b>Chapter 1: First Look at Ionic</b>	<b>141</b>
The beginning	141
Apache Cordova	143
What is Ionic?	145
Summary	151
<b>Chapter 2: To-Do List App</b>	<b>153</b>
Creating our first application	153
The Ionic workflow	154
Summary	161
<b>Chapter 3: Running Ionic Apps</b>	<b>163</b>
Running our todo app	163
Summary	171
<b>Chapter 4: Ionic Components</b>	<b>173</b>
Creating a new to-do list application	173
Summary	189
<b>Chapter 5: The London Tourist App</b>	<b>191</b>
Introduction to the London Tourist App	191
Summary	206
<b>Chapter 6: Advanced Ionic Components</b>	<b>207</b>
The Ionic Popover	207
The Ionic Modal	215
Summary	220
<b>Chapter 7: Customizing the App</b>	<b>221</b>
Customizing the look and feel of your app	221
Ionic SCSS overview	222
\$ionicConfigProvider	233
Summary	233
<b>Chapter 8: Building a Simple Social App</b>	<b>235</b>
The Ionic tabs application	235
The <ion-tab> element	241
Summary	250
<b>Chapter 9: Connecting to Firebase</b>	<b>251</b>
Extending our tabs-app Ionic app	251
Firebase	255
Summary	263

*Table of Contents*

---

<b>Chapter 10: Roundup</b>	<b>265</b>
Uncovered features of Ionic	265
Useful resources	271
Summary	272
 <b>Module 3: Ionic Cookbook</b> <hr/>	
<b>Chapter 1: Creating Our First App with Ionic</b>	<b>275</b>
Introduction	275
Setting up a development environment	276
Creating a HelloWorld app via CLI	278
Creating a HelloWorld app via Ionic Creator	281
Copying examples from Ionic Codepen Demos	285
Viewing the app using your web browser	287
Viewing the app using iOS Simulator	290
Viewing the app using Xcode for iOS	291
Viewing the app using Genymotion for Android	294
Viewing the app using Ionic View	301
Customizing the app folder structure	305
<b>Chapter 2: Managing States and Navigation</b>	<b>307</b>
Introduction	307
Creating a tab interface with nested views	308
Creating a multistep form with validation	316
<b>Chapter 3: Adding Device Features Support</b>	<b>331</b>
Introduction	331
Taking a photo using the device camera	332
Capturing video and allowing playback	338
Composing an email with an attachment from an app	342
Picking and adding a contact	347
Adding Google Maps with geocoding support	355
<b>Chapter 4: Offline Data Storage</b>	<b>369</b>
Introduction	369
Creating a to-do app using ngStorage for Local Storage	370
Creating a social networking app using SQLite	377
<b>Chapter 5: Handling Gestures and Events</b>	<b>393</b>
Introduction	393
Detecting drag events with a gesture coordinate	394
Communication between a view, controller, and directive using events	400

---

*Table of Contents*

<b>Chapter 6: App Theme Customization</b>	<b>409</b>
Introduction	409
Customizing themes for specific platforms	410
Creating an introduction screen with a custom header	413
<b>Chapter 7: Extending Ionic with Your Own Components</b>	<b>423</b>
Introduction	423
Creating a scroll progress bar directive	424
Creating a custom filter	429
Animating an app using requestAnimationFrame with event binding	434
<b>Chapter 8: User Registration and Authentication</b>	<b>443</b>
Introduction	444
Configuring a Facebook app with Firebase authentication	445
Configuring a Twitter app with Firebase authentication	450
Configuring a Google+ project with Firebase authentication	453
Creating an Ionic social authentication project for Facebook using \$firebaseAuth	457
Creating a LinkedIn app and configuring authentication in Auth0	465
Integrating Auth0's LinkedIn authentication in an Ionic project	478
<b>Chapter 9: Saving and Loading Data Using Firebase</b>	<b>483</b>
Introduction	483
Saving array data to Firebase	484
Rendering a large Firebase data set using collection-repeat	487
Saving form data to Firebase	492
<b>Chapter 10: Finalizing Your Apps for Different Platforms</b>	<b>501</b>
Introduction	501
Building and publishing an app for iOS	502
Building and publishing an app for Android	507
Using PhoneGap Build for cross-platform applications	510
<b>Bibliography</b>	<b>513</b>
<b>Index</b>	<b>515</b>

---



# Module 1

## **Getting Started with Ionic**

Get up and running with developing effective Hybrid Mobile Apps with Ionic



# 1

## All About Hybrid Mobile Apps and Ionic Framework

In this chapter, we are going to learn what a Hybrid Mobile Application is and the current technology ecosystems supporting it. We will also be introduced to Ionic Framework and the reasons that should lead you to decide on Ionic as your preferred choice. The topics covered in this chapter will be as follows:

- Introduction to a Hybrid Mobile Application
- Using web technologies to develop for mobile devices
- What is AngularJS?
- Why use Apache Cordova?
- Introducing Ionic Framework

The term 'website' has become a word of the past. 'App' is the new buzzword, and the world is moving away from old software systems to new jazzy apps. App, or application in terms of a software, is a more sophisticated system, which involves enabling a lot more features to the user rather than just providing static information like a traditional website.

Web apps have a lot of limitations such as requiring Internet connectivity all the time and restrictions on fully utilizing the hardware capabilities of the device on which you are accessing them. Mobile apps, on the other hand, defy all of these limitations and provide an engaging user experience.

Mobiles have emerged as the most popular channel for user engagement. The number of smartphone users is expected to grow to nearly 2.16 billion in 2016 (<http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694>), which is more than one quarter of the global population. The growth rate of smartphone usage has been tremendous and is expected to grow day by day.

Mobile adoption is associated with some of the following trivial points:

- Mobiles are the only device that we keep with us the whole day
- Kids (between the ages of two and five) know more about using a smartphone than tieing their shoelaces
- Time spent on mobile phones is increasing 14 times faster than time spent on desktop media
- People check their smartphones first thing when they wake up instead of wishing good morning to their partners or family

After reading through these points, one must think that the perfect mobile strategy has become a necessity for every business. Going mobile is not a choice any more, rather everyone has to decide which way to go: mobile-first, mobile-only, or mobile-after. If you are a new start-up, an existing enterprise, or an individual developer, Native Mobile apps have a steep learning curve and high development costs for covering all platforms. Mobile ecosystems have become fragmented with multiple OSs such as iOS, Android, Windows, and numerous OEMs such as Samsung, LG, HTC, and others. A Hybrid Mobile Application, about which you will learn in this chapter, is the perfect savior for you. Ionic Framework is a popular hybrid app development framework that helps us in creating Native-looking Apps for multiple platforms using a single codebase.

## **Introducing a Hybrid Mobile Application**

A common misconception is that a Hybrid Mobile Application cannot be installed on the device, but that is wrong. A Hybrid Mobile App is like any other Native Mobile App, which can be installed on devices and published using App stores. They can access the device hardware such as camera, accelerometer, GPS, and so on.

As we have discussed, there are multiple mobile platforms such as iOS, Android, Windows, plus many new ones such as Firefox OS and Tizen that have emerged lately. The development environment and programming languages are different for each of these. We have to code using Objective-C for iOS apps, Java (Android SDK) for Android apps, C#/VB.net with XAML for Windows Phone apps. If any entity requires its mobile presence across all these platforms, multiple teams and different codebases need to be maintained, which is too cumbersome.

Hybrid Mobile Apps can be developed for multiple platforms using a single codebase. However, some specialized code needs to be written for each platform to harness the native APIs for it.

## **Types of Hybrid Mobile Apps**

There are broadly two categories of Hybrid Mobile Apps in the industry:

- WebView-based Hybrid Apps
- Cross-compiled Hybrid Apps

### **WebView-based Hybrid Apps**

Each native mobile platform has a common control/component called a WebView, which is nothing but a Chromeless browser. This component is utilized to open locally hosted web content, for example, HTML pages, CSS files, and JavaScript code. HTML5 and CSS3 provide capabilities to develop responsive apps, which can render nicely on multiple screen sizes. The web technologies have evolved to handle the touch interactions that make them a perfect candidate for developing solutions for smartphones and tablets. The examples of development platform and frameworks using this approach are Cordova, Ionic Framework, KendoUI Mobile, F7, Mobile Angular UI, Onsen UI, and many more.

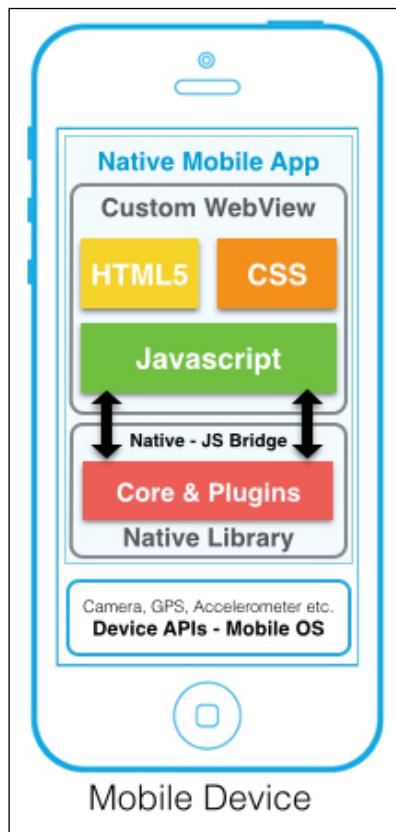
### **Cross-compiled Hybrid Apps**

Another category of hybrid app involves cross-compiling multiple native apps from a single programming language. For example, the developer will code using a single language, say, A, which can be converted at compile time or run time into native language components. Generally, these types of frameworks and platforms leverage creating a bridge or a mapping of native components to their custom constructs in the programming language intended for development. Examples for this category are Xamarin (C#), Kony (JS), Corona (C Lang.), Qt(C++), and many more.

We will be talking about WebView-based Hybrid Apps in this book as they are more suitable for large-scale and complex Mobile Apps. In this next section, we will learn more about the anatomy of Hybrid Apps and how they are capable of developing Native Apps for multiple platforms.

## Anatomy of a Hybrid Mobile App

Hybrid Apps are no different from Native Mobile Apps that are installed on any mobile platform such as Android or iOS. On any platform, the core device APIs for hardware such as GPS, Camera, Accelerometer, and so on will be exposed by the Mobile OS. The following diagram shows the anatomy of Hybrid Mobile App:



These APIs are consumed by the native code of your Hybrid Apps. All the components of Hybrid Apps are discussed in detail in the following section.

## Custom WebView

Each native platform development kit has a component called WebView, which is nothing but a Chromeless browser control. WebView has the capability to open local or remote web content, which is exploited by Hybrid App frameworks to display the UI of the app using web technologies. This is the most important component of a Hybrid App and has a significant role in deciding the performance of the app.

WebViews in popular platforms such as iOS and Android used to have a different rendering engine and JavaScript engine than the latest browser (Chrome or Safari). Last year, Apple released a new control in its SDK called WKWebView, which uses all the performance optimization such as the Nitro JavaScript engine used by the Safari browser on iOS. On similar lines, Google also released an updated WebView, which uses the rendering engine and JavaScript runtime of Chromium (Chrome browser). Google has also launched a new feature called Updatable WebView from Android (5.0) Lollipop, which enables you to upgrade only the WebView of your Hybrid App.

Crosswalk is an interesting open source project that enables app developers to embed custom WebView into your Hybrid App. With these advancements, Hybrid Apps have become capable of using the latest web features such as WebRTC for real-time multimedia communication, and WebGL for advanced graphics rendering.

## Native library

All the Hybrid App development frameworks based on WebView have their own native library. It comprises some basic utility functions that support the Hybrid App such as creating app configs, bootstrap code for Native Apps, customization for the WebView, common error/exception handling logic, and so on. The native library is specific to the mobile platform as it involves interacting with the core OS APIs and components.

The most popular framework, for example, Apache Cordova/Phonegap has an architecture of dividing its native library into a core section and pluggable components called plugins. This helps in reducing the bare minimum size of a Hybrid App. Developers can use only the plugins they require for a specific app. A plugin will include native code for a particular feature and a JavaScript interface exposing the native functionality. For example, if you want to use the Fingerprint Authentication API for iOS, you can just include the plugin for iOS and use it apart from the core. It also enables communities to contribute by developing open source plugins.

## Native to JS Bridge

In a WebView-based Hybrid App, the UI is always written using web technologies, and JavaScript is the language for writing logic and hence we need to call our native code from JS and get results to JS also. A bridge has two functions, one is to enable JS to call any native method and the other is to allow native methods to execute callbacks in JS. The bridge comprises different implementations in different platforms to call JS from native. For example, in Android, Java objects are marshalled into the WebView and can be called from the JS. In iOS, JS calls a specific URL scheme, which is interpreted by the native code. The reverse bridge is a simple global JS function that is called by the WebView passing special arguments such as callback results or specific commands.

## Using web technologies to develop for mobile

After understanding what is happening inside a Hybrid App, it is important to know how web technologies are used to develop Mobile Apps. We can use simple HTML5, CSS, and JS to create mobile-specific UIs and enable them to be viewed in the WebView discussed previously. But any website, even if it is a mobile web app, should not be directly packaged into a Hybrid App. This is the most popular mistake developers make and then complain about the performance of the Hybrid App.

A Hybrid Mobile App UI needs to have proper separation of concerns and can be best developed using **single page architecture (SPA)** or MV\* architecture. It helps in providing a seamless user experience and provide a Native App such as engagement. It also equips developers with segregated areas to code, for example, writing views using HTML5 markup templates, styling using CSS, and logic in JS.

Ionic Framework uses an open source MV\* framework called AngularJS to build robust Native-looking Hybrid Mobile Apps. AngularJS is an extensive topic that cannot be covered here, but we will learn about some basics that are essential in utilizing Ionic Framework to its full potential.

## What is AngularJS?

AngularJS is a JavaScript-based MV\* framework that provides a strong backbone to scalable and complex web apps. It also enables developers to extend HTML and program their apps in a declarative paradigm in lieu of an imperative programming style. AngularJS provides us with a way of creating reusable components, setting standard templates in HTML, and reusable business logic with the ability to bind data dynamically to it.

AngularJS is a perfect fit for creating rich Mobile UI Apps as it provides a robust structure to the frontend, which is a reason why the Ionic team has chosen it as their core.

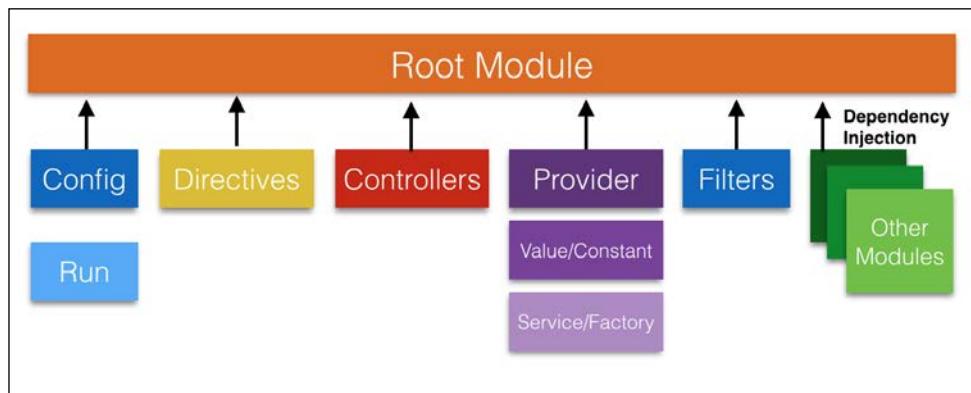
## Important concepts in AngularJS

In order to build apps using AngularJS we need to understand the core concepts used in AngularJS and learn how to use them. The prerequisite for learning AngularJS is decent knowledge of HTML, CSS, and JavaScript. The core concepts that will be discussed include modules, directives, controllers, expressions, and filters.

## Modules

In AngularJS, modules are at the core of everything because an AngularJS App is defined as a module itself. A module is a container for different sections of the app such as controllers, directives, services, and so on. A module can have other module dependencies injected at run time. This feature of AngularJS is called **DI (Dependency Injection)**. It provides super flexibility for unit testing as dependencies can be mocked and injected.

Each module has two important lifecycle hooks implemented as methods registering callbacks. The methods are `config` and `run`. The `config` method is used to set up or provide important configuration settings such as routes, states, and so on, whereas the `run` function is used like the `main` method for initiating the module inside the callback registered:



## Directives

Directives are the most important and yet the most complex part of AngularJS. They can be easily described as a set of markers on DOM elements such as element name, CSS class, an attribute, or a comment, which lets the AngularJS compiler know that specified behavior needs to be attached there. It is advised to encapsulate any DOM manipulation logic into a directive while developing an AngularJS App.

There are plenty of in-built core directives that are part of the `ng-module` and used in each angular app. We will discuss the essential ones in order to understand the functioning of a directive.

`ng-app` is the core directive that bootstraps our app. The root angular module name needs to be passed to this directive and is generally used as an attribute, for example:

```
<html ng-app="my-app">
```

`ng-model` is a directive used for binding models from controllers to the views. We will learn about the scope in the text ahead, which is used to hold the models as Plain Old JavaScript Objects. `ng-model` is used with input, select, and text area controls, for example:

```
<input type='text' name='textField' ng-model="my-var">
```

There are many other directives such as `ng-class`, `ng-show`, `ng-hide`, and so on, which you will use while developing your app. Ionic Framework has built most of its components as custom directives and will be used frequently to develop Hybrid Apps using the framework.

## Controllers

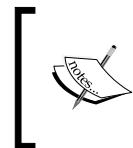
In AngularJS, a controller is a constructor function used to augment the view models. The controller can be initiated in two ways, either using the directive `ng-controller` or it can be associated with a route/state. According to Angular docs, controllers should be used for setting up the initial state of the `$scope` (view model) object and adding behavior to it.

We should refrain from using it for the following logic:

- DOM manipulations
- Formatting input
- Input validations
- Sharing of data should be done using services

The example code for a basic controller is:

```
var newApp = angular.module('NewApp', []);
newApp.controller('FirstController', ['$scope', function($scope) {
    $scope.modelObj = { name:'MyDummyObject' };
    $scope.updateName = function(newName) {
        $scope.modelObj.name = newName;
    }
}]);
```



We define the dependency injection twice, first as a string and then as an argument to avoid problems during the minification of your JS files. Read more details at <https://docs.angularjs.org/guide/di>.



## Services

In AngularJS, services are used to store business logic and organize your code into logical units or entities. Angular services are lazily loaded and are wired together using Dependency Injection, as discussed earlier. AngularJS services are singletons and thus instantiated only the first time they are encountered as a dependency to any controller or module.

AngularJS has multiple built-in services out of which `$http` is the most important one. The `$http` service provides a wrapper on the browser's `XMLHttpRequest` object, or more popularly known as Ajax requests.

We can create AngularJS services using two module factory methods. The methods used are `.service` or `.factory`. The former is used when we create the service instance using a constructor function, and the latter is used when a factory function returns the instance of the service. Conceptually, we should use `.service` when we are integrating to any external API and use `.factory` if we are creating objects representing app models.

The code sample to create services using both methods is given as follows:

```
// Using Factory Method
var newApp = angular.module('NewApp', []);
newApp.factory('MyService', [function() {
    var serviceInstance = {};
    var privateMethod = function() { return 'result'; };
    serviceInstance.exposedMethod = function() {
        return privateMethod();
    };
    return serviceInstance;
}]);
```

In the given code, a factory function available on the `angular.module` object is used to return a service instance that contains public methods. The service can encapsulate private methods to include logic that should not be exposed:

```
// Using Service Method
var newApp = angular.module('NewApp', []);
newApp.service('MyService', [function() {
    var privateMethod = function() { return 'result'; };
    this.exposedMethod = function() {
        return privateMethod();
    };
}]);
```

Services in Angular can be used in the following ways:

- Representing business entities/models
- Sharing data across controllers
- Interface to external web service calls or Ajax requests

## Templates

In AngularJS, templates are associated with a route/state to display HTML elements and Angular-specific elements. Template code can be directly passed in JS or TemplateURL, for example, URL to the template file can be passed to any object. Angular combines the template to the controller and models to display dynamic content to the user on the browser. `$scope` is used to bind the controller to the template.

Templates can use Angular directives and expressions that are compiled by the `$compile` service. The following code shows a simple Angular template:

```
<html ng-app>
  <head>
    <title>My First Angular Template</title>
  </head>
  <body>
    <h1>Main Section</h1>
    <div ng-controller='MyCtrl'>
      <p>{{ contentStr }}</p>
      <p>Date : {{ dateStr | dateFormat }}</p>
    </div>
    Name: <input type='text' ng-model='name'>
  </body>
</html>
```

## Expressions

Expressions are code snippets put in AngularJS templates to create bindings between controllers and templates. The expressions are, by default, represented by `{{ }}` in the templates. Angular expressions are different from JavaScript expressions as they have some restrictions. They can contain basic arithmetic or concatenation operations, but no control flow statement or function declarations. Angular expressions run against the context of the scope, for example, variables in the bindings are evaluated on the scope object for each respective controller.

## Filters

Filters are used in expressions to format data before displaying it. Filters are applied using the `|` operator. There are in-built filters such as currency, number, date, lowercase, and so on.

Example of expressions with a filter in a template:

```
<p> Total : {{ amount * 32 | currency }} </p>
```

`amount` is a scope model, using the `*` arithmetic operator and the `currency` filter.

## Why use Apache Cordova?

Apache Cordova is a WebView-based Hybrid App development framework used to build cross-platform Native Apps. It is one of the most popular frameworks that has been open sourced by Adobe and is maintained by the Apache Foundation. Adobe maintains another branch with added features named Phonegap. They also have a cloud-based service called Phonegap Build (<http://build.phonegap.com>), which generates the native builds on the fly so that you do not need to install native SDKs.

It follows the same architecture discussed previously, having a minimal core and ability to add plugins for extra functionality to your app. Apache Cordova has a high number of open source plugins that provide excellent capability to Hybrid Apps. Any developer can also create a custom plugin to expose unique native functionality in a Hybrid App. It also consists of a CLI interface to provide commands for managing plugins and automating builds for multiple platforms. Apache Cordova is a widely tested and accepted framework and is recommended for building Hybrid Mobile Apps from web content.

## Introducing Ionic Framework

Ionic Framework is a Hybrid App development framework that enables developers to build Native-looking Mobile Apps using web technologies (HTML5, CSS3, and JS). Ionic Framework is completely open source so that developers can build and publish their apps to the marketplace without any cost.

Ionic is built on top of the AngularJS framework and uses Apache Cordova for building apps from web content. Ionic Framework includes a set of amazing Angular directives that makes it very easy to develop for mobile. For example, ListView, Optimized Touch gestures, Side Menus, Popup, Tabs, and mobile-specific input elements.

Ionic has ready-made UI for mobile components, which helps in rapid application development for Hybrid Mobile Apps. Ionic has native-looking stylesheets for Android and iOS, which automatically get applied based on the platform build.

Ionic Framework has evolved into an ecosystem with a suite of mobile development tools along with the framework itself. Ionic CLI has amazing options such as Ionic Lab and Live Reload, which helps developers save lot of development time. Ionic View is a Native App for iOS and Android where developers can deploy and test their apps on the fly without packaging. Ionic.io is a complete cloud-based backend service platform where developers can manage their app data, view analytics, and manage push notifications from a single console.

## Summary

We have learnt all about Hybrid Mobile Applications and how easy it is for a web developer to start building Mobile Apps using Ionic Framework. The concepts mentioned in this chapter will suffice in building apps using AngularJS, Cordova, and Ionic. In the coming chapters, we will build amazing Hybrid Mobile Apps that can be deployed to public app stores. In the next chapter we will learn about setting up the development environment and starting an initial project to bootstrap our development journey.

# 2

## Setting up the Environment the Right Way

In this chapter we will learn how to set up our development environment and build a dummy app to verify the correct setup. This chapter also defines the possible and popular issues faced during setup so that you do not face any bottlenecks during the process. The instructions are separated for Windows OS, generic Linux OS, and Mac OS.

As we have discussed, Ionic has evolved into an ecosystem from a basic framework. It is very important that we set up our development tools the right way to leverage the full potential of this ecosystem.

We will be setting up a lot of software just for developing Mobile Apps, but if we want to use the power of developing for multiple platform apps using a single codebase, we have to do this. Any Native App is always built using the Native Development tools and hence we have to install and set up a Native Mobile development environment for all the platforms.

Apart from the build tools of native environments, a dev environment should provide ease of use and speed to the developer. Installing Ionic CLI based on NodeJS and Cordova would provide the extensive power of bootstrapping Ionic Projects with starter templates, managing Cordova plugins, and other dependencies. Code Editor should provide the developer with complete support for the programming languages an app is being built in. We will learn about some popular Code Editors that gel with Ionic App development. We will also build a dummy app to gain confidence of starting the actual development.

The following topics will be covered in this chapter:

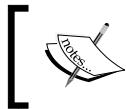
- Setting up Native Mobile development environments:
  - For iOS
  - For Android
  - For Windows phone
- Ionic Development Environments:
  - Installing NodeJS
  - Installing Cordova and Ionic CLI
- Basic commands in Ionic CLI
- Building a dummy app
- Popular issues faced and solutions
- An alternative to the installation fuss—Ionic Play
- Using Ionic with different Code Editors:
  - Brackets
  - Sublime Text
  - Visual Studio

## **Setting up Native Mobile development environments**

Native Mobile development environments are specific to your operating system and hence there will be separate instructions for installing on Windows/Mac/Linux. The native environment SDKs are heavy so would require fast Internet speed to download. You need to be on a good network to be able to download it.

### **For iOS**

These days iOS Apps are built for iPhone, iPad and iWatch devices. Objective-C is the primary language used to develop iOS Native Apps. Xcode is Apple's **IDE (integrated development environment)**, which includes a graphical user interface and many other features. iOS SDK is required along with Xcode as it provides additional tools, compilers, and frameworks to build iOS Apps for phones, tablets, and smartwear devices (iWatch).



For developing or building iOS Apps, a Macintosh machine is strictly required so if you do not have a Mac, please skip this section. Mac machines should have OS X 10.9.4 or later.

In order to download the latest version of Xcode and iOS SDK (bundled) for free, please follow these steps:

1. Go to the **Apple App Store** on your Mac (search for it or open from the dock).
2. Search for Xcode software to download in the top-right corner search box.
3. Click on the **Free** button to download it.

The website link for downloading Xcode is <https://developer.apple.com/xcode/>.

Xcode will be downloaded and installed in the Applications folder of your Mac machine.

## **Installing and running simulators for testing**

In order to test our iOS apps, we either require a physical device or we can test it on a simulator. By default, the simulators are present in the Xcode. In order to install different iOS simulators for specific OS versions, please go to <https://developer.apple.com/ios/download/>.

To run the simulator, open Xcode, right click on the Xcode icon in the dock and go to **Open Developer Tools | iOS Simulator**. It will open an iOS device such as an iPad or iPhone, which can be changed by going to the top menu option **Hardware | Device | Specific Device**.

## **For Android**

Android Apps are developed using a Java programming language so installing a Java environment is a requirement for it. Please download the latest JDK 7 (JRE would not work) from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. It is required to set the PATH and JAVA\_HOME variable to refer to Java and javac binaries (we explain later how to do it).

Apache Ant is a Java build system used by Cordova/Ionic and Android SDK. In order to install Ant properly, download the binary from the link <http://ant.apache.org/bindownload.cgi> and follow these steps:

1. Move the downloaded file to a new location/folder for Ant.
2. Unpack the zip file where you want Ant to be installed.
3. Set the `PATH` variable and `ANT_HOME` to this directory.

Android SDK is available in two variants, one is a standalone SDK and the other is bundled with Code Editor Eclipse or indigenous Android Studio. There are separate binaries for Windows, Linux, and Mac OS. It is recommended to download and set up the standalone SDK for developing Hybrid Apps unless you want to try out native Android development too. We will be discussing setting up the standalone Android SDK only.

## Download links

The following are some downloading links:

- Standalone SDK (recommended) [for all OSes]: <https://developer.android.com/sdk/installing/index.html?pkg=tools>
- Android Studio bundle [for all OSes]: <https://developer.android.com/tools/studio/index.html>
- Eclipse and ADT bundle [for all OSes]: <http://developer.android.com/sdk/installing/installing-adt.html>

## Setting the environment variables

For Cordova command-line tools to work, or the CLI that is based upon them, you need to include the SDK's tools and platform-tools directories in your `PATH`.

## For Windows

The following are the steps to set up Native Mobile development environments on Windows:

1. Click on the **Start** menu, right-click on **Computer**, and then select **Properties**.
2. Select **Advanced system settings** from the column on the left.
3. Select the **Environment Variables** | **PATH** variable and click **Edit**.

4. Append the path with a path to the platform tools and tools folder of SDK.

For example,C:\Development\adt-bundle\sdk\platform-tools;C:\Development\adt-bundle\sdk\tools

## For Linux/Mac OS

On a Mac you can use a text editor and on a Linux you can use vi editor to create/modify the `~/.bash_profile` file, adding a line such as the following, depending on where the SDK installs:

```
$ export PATH=${PATH}:/Development/adt-bundle/sdk/platform-tools:/Development/adt-bundle/sdk/tools
```

Add the paths for Java and Ant if needed. This line in `~/.bash_profile` exposes these tools in newly opened terminal windows. If your terminal window is already open in OS X, or to avoid a logout/login on Linux, run this to make them available in the current terminal window:

```
$ source ~/.bash_profile
```

## Managing Android SDK and emulators

Android has a large fragmentation in the OS versions being used and has different SDK packages for various versions. In the SDK tools, there is an SDK manager that helps to manage different SDK API versions. In order to manage SDK Manager, open `SDK Manager.exe` in Windows, whereas on a Mac/Linux open a terminal and navigate to the `/tools` directory in the location where the Android SDK is installed, then execute `android sdk`.

In Android, to test your apps, install it on an actual physical device or create a new emulator/virtual device using the SDK tools. These next steps should be followed:

1. Open the command prompt in Windows and terminal in Linux/Mac.
2. Go to the location where Android SDK is installed under the `tools` folder.
3. Execute the command `$ android avd` to open the **Virtual Device/Emulator Manager**.

## **Alternative to Android emulators – Genymotion**

Android emulators are known to be slow and sloppy. If you want to debug or test your apps on emulators, use the Genymotion emulator as it is faster than the Android emulators. Apart from being fast, it has 20 mobile devices preconfigured to emulate. The Genymotion emulator uses VirtualBox, so install the latest version and then install Genymotion. The link to the Genymotion website is <https://www.genymotion.com/>.

## **Ionic development environment**

Ionic Framework is based on Apache Cordova, which takes care of build and plugin management. Apache Cordova CLI uses the NodeJS package manager called NPM for dependency management.

## **Installing NodeJS**

NodeJS is a JavaScript-based server-side environment to build backend systems. NodeJS has an efficient package manager that takes care of installing and maintaining a central repository of packages, version management, and dependency management. NPM has become the default dependency manager for a lot of other frameworks apart from NodeJS. Ionic Framework also leveraged this successful package manager so we have to install it for Hybrid App development.

In order to install NodeJS, download the binary for the respective OS – Windows, Linux, or Mac OS from <https://nodejs.org/download/>.

On a Mac or Linux, you can use a package manager such as brew to install NodeJS:

```
$ brew install node
```

After the node installation is complete, please type in the following commands to test the installation:

```
$ node -v  
$ npm -v
```

The appropriate installed version for node and npm will be written to the output.

There is a config.json file generated that stores the dependent modules and meta information about the specific NodeJS project.

## Basic npm commands

In order to install any npm package, the `install` command is used. It has an optional flag `-g` for installing any package globally so that it is accessible from anywhere. All the CLI packages such as Cordova CLI and Ionic CLI will be installed using these commands only. In order to update a package, run the second command:

```
$ npm install -g <package_name>
$ npm update -g <package_name>
```

## Installing Cordova CLI and Ionic CLI

Cordova will be installed using the npm utility discussed previously. The Cordova CLI includes a set of scripts to automate the Cordova build process for wrapping of web content into Native Mobile Apps. It also includes commands to manage plugins. As discussed in *Chapter 1, All About Hybrid Mobile Apps and Ionic Framework*, Cordova is based on an architecture where the core library is included by default, but extra functionality needs to be utilized using plugins.

### Installation command for Cordova CLI

In order to install Cordova, use the npm global `install` command as follows:

- **On a Mac/Linux machine:** `$ npm install -g cordova`
- **On a Windows machine:** `C:\>npm install -g cordova`

Cordova installation can be confirmed by checking the version of the installed package as follows:

```
$ cordova -v
```

The Ionic command-line interface provides an easy-to-use interface to perform basic functions such as creating an app, adding a platform, and building apps. It also provides commands to use Ionic platform services helping in rapid development of Hybrid Apps. Ionic CLI is installed in a method similar to Cordova with the following commands:

- **On a Mac/Linux machine:** `$ npm install -g ionic`
- **On a Windows machine:** `C:\>npm install -g ionic`

Ionic CLI extends the Cordova CLI and hence most of the Cordova commands need not be run directly. They can be run using the `ionic` keyword, where internally the Cordova package will be used. You can also view a list of exhaustive Ionic commands by executing `ionic help`.

## Basic Ionic CLI commands

Ionic commands help you smoothly create a new project and test it easily while you are developing it. The first command you should learn to execute is the `ionic info` command. This would help you to know all about your system environment that you have set up so far. If there are any errors, you can debug and resolve them before beginning the development.

The command and a sample output is given here:

```
$ ionic info
Your system information:
OS: Mac OS X Yosemite
Node Version: v0.12.4
Cordova CLI: 5.1.1
Ionic CLI Version: 1.6.2
Xcode version: Xcode 6.4 Build version 6E35b
ios-sim version: 3.1.1
ios-deploy version: 1.7.0
```

Another quick utility command is to go to the Ionic docs for a specific topic:

```
$ ionic docs <TOPIC>
```

Please go through the commands in this section in order to understand the steps required to build a dummy app to test the environment.

The next command to learn is to start a new project. Ionic has multiple flags/options along with this command to set up your project initially. The main command is as follows:

```
$ ionic start [OPTIONS] <PATH> [template]
```

The options are many, including `-appname` to set the name, `--no-cordova` to create a structure without Cordova (for cloud builds such as a phonegap build service), `--template` for selecting the initial template, and so on.

If you want to preview your app in a browser during the development phase, this is the most important command for it. It has a handful of useful options for setting a port number, opening a specific browser, printing console logs, and so on:

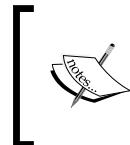
```
$ ionic serve [OPTIONS]
```

In order to add a platform, please run the following command:

```
$ ionic platform [OPTIONS] <PLATFORM>
```

After adding the platforms we want to build, we can run our app on an actual connected device using this command:

```
$ ionic run [OPTIONS] <PLATFORM>
```



In order to run an app on an Android device, enable **Developer Options** and check the **USB Debugging** option in the settings. More info is available at <http://developer.android.com/tools/device.html>.



We can make the app build for all the platforms in one go or a specific platform by mentioning its name in the following command. It builds the apps locally using the native SDK tools:

```
$ ionic build [OPTIONS] <PLATFORM>
```

Hybrid App development relies on the web view or the browser in the native SDK of the platform. Ionic provides a way to change the browser of a specific app, for example, using the crosswalk browser. There is a command to list the available browsers:

```
$ ionic browser list
```

Add a new browser for any platform and also have the ability to revert back:

```
$ ionic browser add crosswalk  
$ ionic browser revert <PLATFORM>
```

Or else you can use the following command:

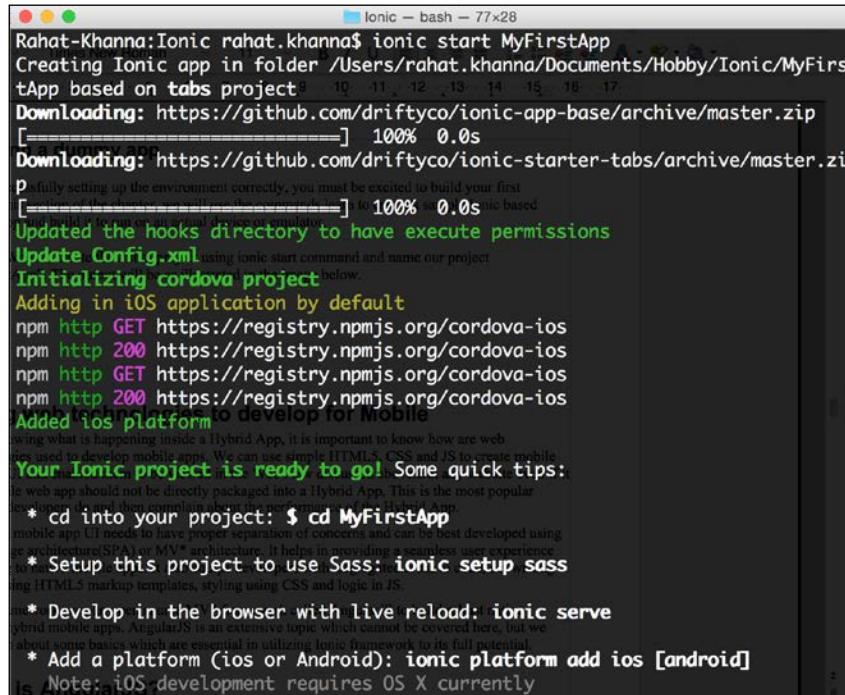
```
$ ionic browser rm crosswalk
```

Ionic provides a way to list and manage extra add-ons or browser packages to your app using the Ionic CLI, which we will learn in future chapters.

## Building a dummy app

After successfully setting up the environment correctly, you must be excited to build your first app. In this section of the chapter, we will use the commands to learn how to create a sample Ionic-based Hybrid App and build it to run on an actual device or emulator:

1. We will create the initial project using the Ionic start command and name our project MyFirstApp. The output will be as illustrated in the following screenshot:



The screenshot shows a terminal window titled "Ionic - bash - 77x28". The output of the command "ionic start MyFirstApp" is displayed. It shows the creation of an Ionic app in the folder "/Users/rahat.khanna/Documents/Hobby/Ionic/MyFirstApp" based on the "tabs" template. It includes steps for downloading dependencies from GitHub, updating permissions, initializing a Cordova project, adding an iOS platform, and providing tips for development. The terminal shows the progress of the download and the final message: "Your Ionic project is ready to go! Some quick tips:".

2. After the successful creation of your project with the default Ionic template, go to your project folder using the command `$ cd MyFirstApp`.

Now, as we are in an Ionic project folder, we can run Ionic project-specific CLI commands. Any platform can be added to the project to build the specific app:

```
$ ionic platform add android
```

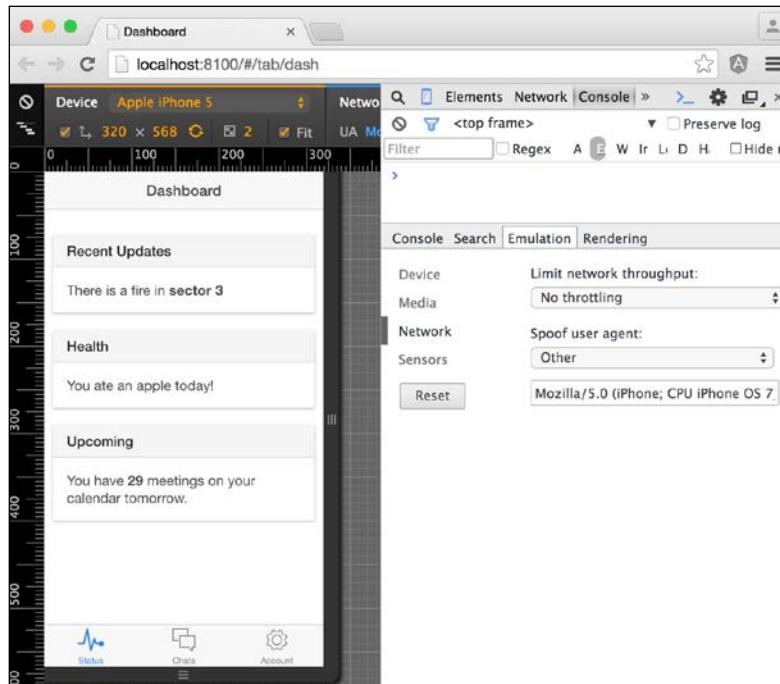
We will add an Android platform as it will work on all machines, such as Windows, Linux, or Mac. Ionic will download Android-specific default resources such as icons and splash screens, and also add some important plugins by default. A customized keyboard plugin is bundled with the Ionic app.

3. In order to test the app and view it in your browser, use the Ionic serve command:

```
$ ionic serve
```

This will open the Ionic app in your default browser.

The latest Chrome version provides a way to emulate various mobile device screen sizes. Open **Developer Tools** (*F12*), select **Console** from the top-right corner, and select **Emulation**. You can alternatively select the mobile icon from the top-left corner of the window. Please see the following screenshot:



4. Now, we can build the Android App and run it on an actual device or emulator:

```
$ ionic run android
```

This will fire up the emulator if there is no connected device, then start the build process. It will generate the .apk file and then deploy it to the emulator or the connected device that is available.

This would be a Eureka moment, seeing your first actual app on the device and playing with it. In the next section we discuss some of the common issues you may have faced during the process and provide solutions for them.

## Popular issues faced and solutions

The development environment set up for Ionic Hybrid App development is a lengthy and cumbersome process. It is evident that you may face some issues in installing and setting up all these systems. This section mentions the most common and popular issues faced along with their solutions so that you do not get stuck somewhere:

- **Permission issue [Mac or Linux]:** A lot of the time, the user you are logged in as does not have permission to alter files and directories while creating projects. This problem can occur in npm `install` commands and Ionic `start` commands.
  - **Solution:** Use a `sudo` keyword before any command to run in privileged mode.
- **Unable to find npm global modules:** If you have installed a global npm package using `-g` flag but are unable to use it, you may need to add the npm directory to your path in order to invoke globally installed modules.
  - **Solution:** Add the npm directory location to your path. On Windows, npm can usually be found at `C:\Users\username\AppData\Roaming\npm`. On OS X and Linux it can usually be found at `/usr/local/share/npm`.
- **Git command-line tool not installed:** For Cordova/Ionic plugins to work properly, it fetches code from git repositories.
  - **Solution:** Download and install git from <https://git-scm.com/downloads>.

- "**Failed to run 'android'" or "adb command not found"**: This means that the Android PATH has not been correctly set for this session. Please go to the *Installing Android* section and read how to set the path.
  - **Solution:** Set the PATH variable correctly.

If you face any other issues, please research online to find possible solutions for them.

## **Alternative to installation fuss – Ionic Playground**

Ionic Creator Drift Inc. has provided an excellent cloud service called Ionic Playground to build Ionic Apps and fiddle around with Ionic Framework. Ionic Playground is similar to JSFiddle where you can write/edit small Hybrid Apps using Ionic Framework with live preview. It also has the capability to save your app and resume work later using a unique URL. The URL can be shared with multiple people and anyone can fork it to add their own flavor to it. The URL is <http://play.ionic.io>.

## **Using Ionic Framework with different Code Editors**

Code Editors are an integral part of a development environment and hence it is important to choose the perfect editor that augments the coding style for Ionic. Here we suggest some Code Editors that support Ionic Framework very well.

### **Brackets**

Brackets is a new but promising Code Editor for building JS and HTML5 Apps as it is built using the same technologies. It has the capability to add extensions to augment its features. The Code Editor can be downloaded from <http://brackets.io>. Brackets has an excellent extension for Ionic, which can be found at <http://ionicbrackets.com>.

## Sublime Text

Sublime Text is the most popular choice among all web developers. It is a very lightweight and fast Code Editor, with full support for JavaScript and HTML5. It also has the capability to add plugins. Ionic also has a plugin for Sublime. Download the Code Editor from <http://sublimetext.com/download>.

## Visual Studio

Visual Studio has recently launched support for Ionic by providing starter project templates directly available in the menu itself. You can install them in your Visual Studio after installing Cordova tools. The Ionic Project templates installation can be accessed by going to **Tools | Extension and Updates**, and in the **Online** tab searching for **Ionic**.

## Summary

We have learned how to set up the complete development environment for building Hybrid Mobile Apps using Ionic Framework. We have set up the platform environments for iOS and Android. We have also successfully created a dummy app and tested it. In the next chapter we will learn the different templates to bootstrap and the underlying architecture and structure of an Ionic App.

# 3

## Start Building Your First Ionic App

In this chapter, we will build our first Ionic App and build it to test on actual devices. Starting a project seems to be the toughest step but if done well, eases the whole process. It is rightly said, "well begun is half done," so we will learn how to start our project perfectly and create the skeleton for our project. All the options of the starter templates available with Ionic Framework will be explained in detail. In this chapter, we will learn about Ionic Framework while building a sample e-commerce Mobile App. In this chapter, we will discuss the design and architecture for our app. The topics covered in this chapter are as follows:

- Starting a new project:
  - Multiple ways to start a new project
- The anatomy of an Ionic Project:
  - The project's folder structure
  - Main components
- Ionic starter templates:
  - Blank
  - Tabs
  - Side menu
  - Maps
- Design and architecture for an e-commerce sample app – BookStore

In Ionic Framework, starting a project has been made very easy, and using the library is straightforward too. Ionic Framework is perfect for those people who are individual developers looking to launch their Mobile App into the market, or techie entrepreneurs who are planning to start a mobile business. Ionic Framework has a smooth learning curve and reading this book can be used to develop a production-ready Hybrid Mobile App.

All the code samples in the subsequent chapters will be streamlined and organized around building the e-commerce Mobile App talked about in this chapter. In this journey, anyone can learn how to take your ideas from the design phase to market launch.

In this chapter, we will also learn in detail how to test our apps in browsers or emulators. Mobile Apps are very cumbersome to debug as we cannot see the output of console logs from the actual devices or emulators. We will learn about various methods to aid this process and help to resolve multiple issues arising during the development phase.

## Starting a new project

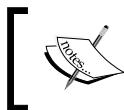
Ionic Framework comprises two major parts, one is the library consisting of JavaScript and CSS files, and the other is the CLI tools and cloud services provided for rapid application development. Ionic CLI has already been set up using the installation steps given in the previous chapter. An Ionic App project can be started in multiple ways with or without the installation of the CLI tools and also using a cloud-based service called **Ionic Creator**.

## Multiple ways to start a project

There are various methods to start a project as shown in the following sections.

### Method 1 – using CDN-hosted library files

An Ionic App majorly requires the Ionic library JS and CSS files for development. These can be referenced directly from the CDN so that our app always uses the latest versions. We do not even require to set up our local environment for this method.



This method should only be used for developing Mobile Apps using Ionic. To develop Hybrid Mobile Apps you would need to use Ionic CLI.



Ionic library is built on top of the AngularJS framework and hence we need to include its library files too. Ionic library includes two types of JS/CSS files, ones that have AngularJS code bundled to it and the others where the AngularJS code is segregated and needs to be included separately. The link to the Ionic CDN website where the URLs for each of the library files mentioned is <http://code.ionicframework.com/>.

We have to include the JavaScript file and the CSS file in a new `index.html` file, which will be the starting point for our Ionic App. A sample HTML code for the `index.html` would look like the following:

```
<!DOCTYPE html>
<html ng-app="FirstApp">
  <head>
    <title>First Ionic App</title>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1,
maximum-scale=1, user-scalable=no, width=device-width">
    <link rel="stylesheet" type="text/css"
      href="http://code.ionicframework.com/1.0.1/css/ionic.min.css">
  </head>
  <body>
    <h1>Bare Minimum App</h1>
    <script type="text/javascript"
      src="http://code.ionicframework.com/1.0.1/js/
      ionic.bundle.min.js"></script>
    <script type="text/javascript">
      angular.module("FirstApp",['ionic'])
        .config(function() {
        })
        .run(function($ionicPlatform) {
        });
    </script>
  </body>
</html>
```

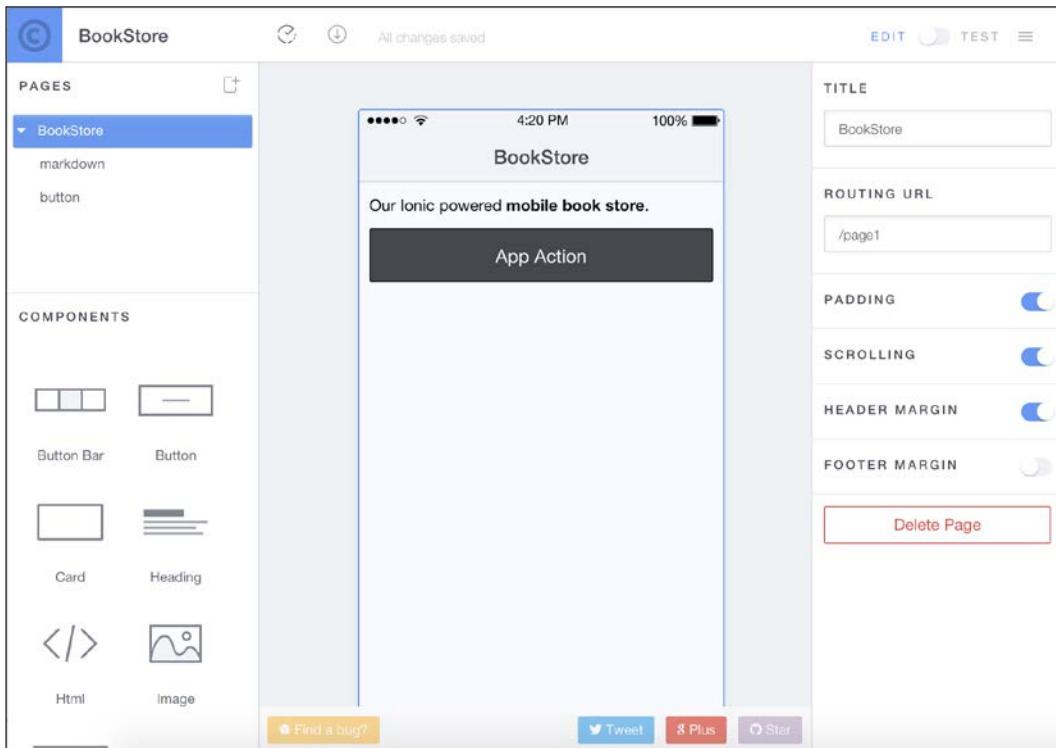
The preceding code is just for starting your app. You need to organize your app-specific JS and CSS code into separate files accordingly as we proceed. Also, in order to view mobile-specific content you can add some Ionic specific directives or code to the preceding HTML file to test the correct output. Please replace the `h1` tag in the file with the following code:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Ionic Blank Starter</h1>
```

```
</ion-header-bar>
<ion-content>
</ion-content>
</ion-pane>
```

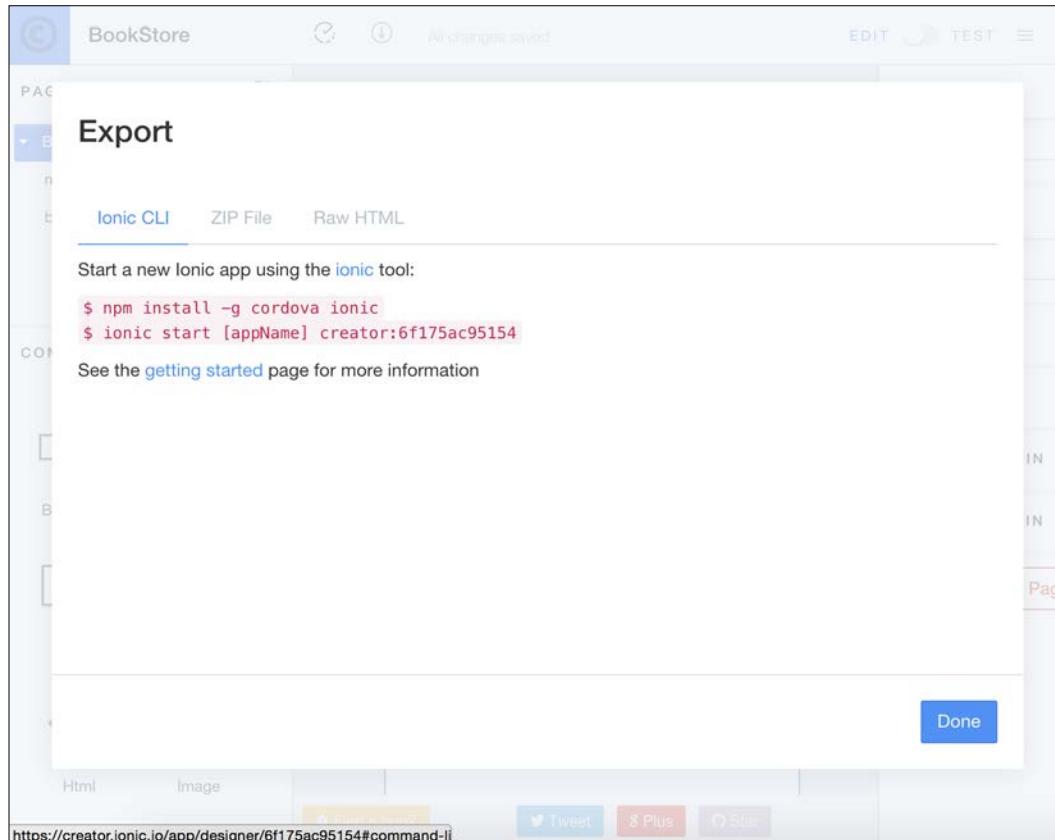
## Method 2 – using Ionic Creator to design a prototype and start a project

Ionic Creator is a drag and drop-based cloud console to design and prototype Ionic Apps. Ionic Creator provides an easy-to-use interface to create views and interactions for your Ionic App. We can link multiple views to create complete use cases and flow. It is the easiest way to start a new project and build your app rapidly. It also contains practically all the Ionic components that can be customized from the UI options themselves as shown in the following screenshot:



After designing your app using Ionic Creator, you can export it to code it further and include complex functionality. The code can be exported in the following ways:

- **Ionic CLI:** Two commands mentioned in the UI with a unique project ID to download.
- **ZIP File:** Downloading the zip file directly using this option.
- **Raw HTML:** An integrated HTML code is generated, which can be copied and pasted. The following screenshot shows the popup view to export and download your Ionic Creator project:



This method is mostly suitable for people who have beginner-level knowledge of HTML and CSS. They can start their project using downloaded content from the Ionic Creator project. The previous screenshot shows exporting the content.

Projects being developed without setting up the local environment and CLI can use a cloud-based service by Adobe called Phonegap Build to build apps. This service can generate iOS, Android, and Windows builds without installing the development environments locally. Ionic also recently launched a new tool called **Ionic Lab**, which can be used for this. We will discuss this further in the last chapter.

## Method 3 – using Ionic CLI locally

Ionic CLI can be used to start a new project using the command used in the previous chapter. Ionic CLI's start command creates a new project with a complete folder structure and sample code from one of the templates chosen in the command. There are multiple flags mentioned in the following table that can be used with the start command:

```
$ ionic start [OPTIONS] <PATH> [template]
```

Options can be set using the flags in the following table:

Flag	Description
[--appname   -a]	Name for your app that is easily readable (use quotes). Example: <code>ionic start -appname "MyApp" myapp blank</code> Or: <code>ionic start -a "MyApp" myapp blank</code>
[--id   -i]	Unique package name set for the app that will be set in the config as <widget_id>: <code>ionic start -id com.myorg.myapp myapp blank</code> Or: <code>ionic start -i com.myorg.myapp myapp blank</code>
[--no-cordova   -w]	Use this flag if you do not want to create a Cordova project: <code>ionic start --no-cordova myapp blank</code> Or: <code>ionic start --w myapp blank</code>
[--sass   -s]	Set up the project to use Sass CSS precompiling: <code>ionic start --sass myapp blank</code> Or: <code>ionic start --s myapp blank</code>
[--list   -l]	List the starter templates available: <code>ionic start --list</code>

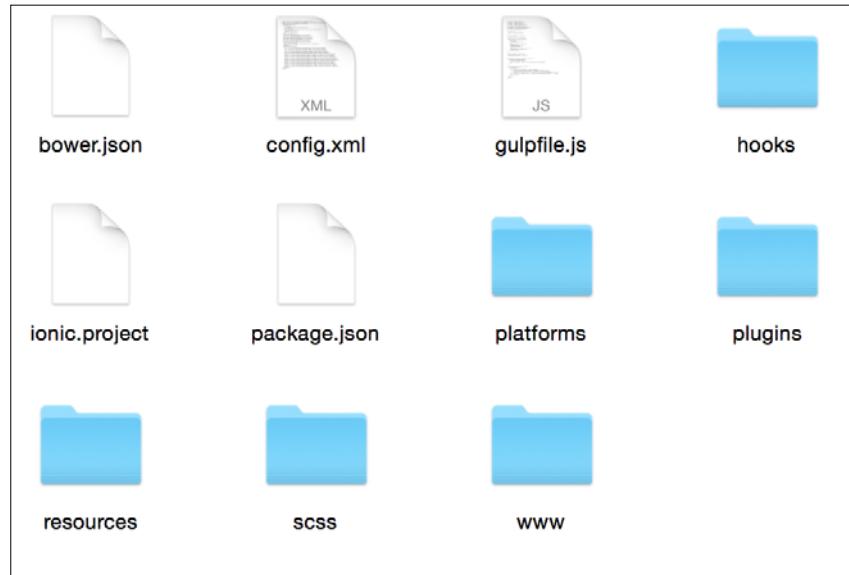
In the next section, we will learn about the different sections and components of the app generated during the start phase of the app using any template.

## The anatomy of Ionic Project

**Ionic Project** is referred to in the project setup using the Ionic CLI tools locally. The later section will describe the main components of Ionic App that are relevant for the projects started using CDN files or Ionic Creator. In order to understand Ionic Project, we need to understand the project structure and the sections in the Ionic App code.

## Project folder structure and important files

The Ionic App project folder structure includes folders for Cordova files, native platform code, and web content code. It also contains configuration files for dependency managers used in an Ionic App as shown in the following screenshot:



We will discuss briefly the important folders and files along with their purpose. These important folders and files are:

- `bower.json` (file): This is used as a configuration file for Bower, dependency managers for frontend libraries, and frameworks. The library directory for storage is written in a separate file `.bowerrc` and is set to `www/lib` by default.
- `config.xml` (file): This file contains the configuration settings for the Cordova project. It is used to define meta information about the app and permissions required for the apps to be installed on specific platforms. It also includes the Cordova plugins used in the app. To read more about the different sections and tags available to be put in `config.xml` see [https://cordova.apache.org/docs/en/4.0.0/config\\_ref\\_index.md.html](https://cordova.apache.org/docs/en/4.0.0/config_ref_index.md.html).
- `gulpfile.js` (file): This is used as config script for Gulp, the build management system used for Ionic web content. It has multiple plugins used for different tasks such as concatenation, auto-installing Bower, and compiling Sass files to CSS files.
- `ionic.project` (file): This file is used to store meta information regarding Ionic Project and associated Ionic.io cloud account services associated with it.
- `package.json` (file): Most of the systems such as Gulp and Bower are installed and managed using Node's package manager NPM, and `package.json` is the file to store versions of the NPM packages installed in Ionic Project.
- `hooks` (folder): This folder contains code for Cordova hooks, which is a way to execute some code during a life cycle event in a Cordova build life cycle. Ionic utilizes the `after_prepare` hook to inject platform-specific CSS and HTML code.
- `platforms` (folder): This folder consists of the platform subfolders that are added to Ionic Project. The subfolders contain native projects for the Ionic App.
- `plugins` (folder): The `plugins` folder contains Cordova plugin subfolders, which include the JS library and native codebase for the specific plugins. The plugins are added and managed using the CLI commands, which we will learn about in future chapters. Cordova plugins can be developed by any developer and can be integrated using a `git url/repo`.
- `resources` (folder): A Mobile App requires static resources such as icons and splash screens. Each platform has specialized requirements regarding the sizes and format for icons. Ionic CLI provides special commands to automatically generate platform-specific resources from generic ones.

- **scss** (folder): scss consists of .scss files based on the SASS framework. SASS is an extension of CSS3, which empowers the developer with multiple constructs to organize CSS code of the Ionic App. It includes features such as nested rules, variables, mixins, and more. The default file `ionic.app.scss` contains important base variables such as theme colors and font paths. SASS is not necessary for developing Ionic Apps but if you know the technology, you can add your scss files to this folder and Gulp will automatically compile them into CSS.
- **www** (folder): This is the most important folder that contains the actual code for Ionic App. It contains the HTML, CSS, JavaScript, and other static files such as images, fonts, and so on. Ionic Project, by default, creates subfolders under this folder for `css`, `img`, `js`, and `templates` (HTML) code.

It is recommended to organize your code into different folders, one for each feature, and include all other necessary files such as controllers, directives, and services in the same folder.

## Main components

The main code lies inside the www folder. The Ionic App at its base is an AngularJS app with an Ionic module injected and multiple directives being used. We will discuss the main constructs and components in a basic Ionic App, which are the most essential.

### The index.html file

The most important file and the starting point of your Ionic App is `index.html`. It should contain important meta tags for the viewport to be able to resize properly on mobile devices. If you have created an Ionic Project using CLI, then you do not need to worry about it. If you are trying to create an Ionic App manually, please include this meta tag in the head section of your `index.html`:

```
<meta name="viewport" content="initial-scale=1, maximum-scale=1,  
user-scalable=no, width=device-width">
```

Also, please check the references to the Ionic bundle script or AngularJS script if the bundled library is not used. Also, please check that `cordova.js` is injected as this reference will be required only after Cordova builds the app, and will give a 404 error during the development phase and testing on browsers:

```
<!-- ionic/angularjs js -->  
<script src="lib/ionic/js/ionic.bundle.js"></script>  
<!-- cordova script (this will be a 404 during development) -->  
<script src="cordova.js"></script>
```

The next important thing is the `ng-app` directive and the value given to the attribute. The `ng-app` directive should be present as an attribute on either body tag or the root HTML tag. This is required to bootstrap the Angular app using the same name in `app.js` while defining the root angular module. The body tag can include any Ionic components. We will discuss some common important Ionic directives that will form the base for an Ionic App.

## App.js and the root module

The next important file is the `app.js`, which will be used to bootstrap the Ionic/Angular app. The name of the module should match the value given to the `ng-app` attribute in `index.html`.

It is also required to inject the `ionic` module dependency into our root module so that we can use Ionic library directives and other constructs. We can inject any other modules into the root module, for example, the services, controllers, or custom directives that we write for our app.

In a Hybrid Mobile App, we have to call native functionalities using the Cordova plugin. It is compulsory to listen to Cordova's device ready event, which lets us know that the native Cordova library has been loaded and our HTML code can call native functionality. Similarly, in Ionic, there is a facility to register a callback in the `$ionicPlatform.ready` event, which fires after the Cordova's device ready event. We should check the status of native plugins or call any native functionality inside this ready event callback. The default code in the `run` block of the root module when you create an Ionic App is as follows:

```
angular.module('starter', ['ionic']) // starter.services etc can be added
.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 &&
        window.cordova.plugins.Keyboard) {
      cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
      cordova.plugins.Keyboard.disableScroll(true);
    }
    if (window.StatusBar) {
      // org.apache.cordova.statusbar required
      StatusBar.styleLightContent();
    }
  });
})
```

Inside the `ready` function we are detecting whether the keyboard plugin is present in our Ionic App, and we are hiding the keyboard accessory bar and disabling the scroll. You can change these values as per your requirements in your app.

## Simple content directives – `ion-content` and `ion-pane`

In a basic Ionic App, the two important directives used to put content in are `<ion-content>` and `<ion-pane>`. Generally, the content is put in a navigation view, but navigation directives will be discussed in the next chapter. In order to create very basic apps, direct content directives can be used in the main file or by using templates for different routes.

### `ion-content`

The `ion-content` directive is used to wrap the content of a view, which can also be configured to use Ionic's custom scroll view, or the built-in scrolling of the web view. It is important to note that this directive gets its own child scope and thus any models inside this directive should be referenced accordingly. Some important attributes that can be used along with this directive are given in the following table:

Attribute [type]	Details
<code>direction</code> [string]	Specifying the direction to scroll. <code>x</code> , <code>y</code> , or <code>xy</code> . Default is <code>y</code> .
<code>scroll</code> [boolean]	It specifies whether to allow scrolling of content. Default is <code>true</code> .
<code>overflow-scroll</code> [boolean]	It specifies whether to use overflow scrolling instead of Ionic scroll. The <code>\$ionicConfigProvider</code> object can be used in <code>config</code> block to set this default.
<code>start-x</code> [string]	Initial horizontal scroll position. Default to 0.
<code>start-y</code> [string]	Initial vertical scroll position. Default to 0.
<code>has-bouncing</code> [boolean]	It specifies whether to allow scrolling to bounce past the boundaries of content. Defaults to <code>true</code> on iOS and <code>false</code> on Android.

## **ion-pane**

The `ion-pane` directive is a simple container for adding content. It just adds a class `pane` to the element in which the contents are encapsulated.

The following is the usage of this directive:

```
<ion-pane>
...
</ion-pane>
```

## **The Ionic starter template**

In the Ionic CLI, you can start a new project using a specific template. There are predefined templates available in the Ionic CLI toolset. They all provide a starting point for developers and also act as a skeleton for your new app. We have already seen the command to start a new project with a specific template earlier on in this chapter. If we want to choose the `tabs` template, then the command would look like the following:

```
$ ionic start TabsDemo tabs
```

We will be discussing the important files in each of the templates.

## **The blank template**

It contains the bare minimum code for an Ionic App. It injects Ionic and Angular dependencies into your `index.html` and creates separate folders for `CSS`, `js`, and `img`. It also provides a basic view code in the body tag of `index.html` to show `ion-pane` and `ion-header` content.

## **The tabs template**

The `tabs` template provides an app layout with the bottom three tabs and multiple views for each tab. It sets up the controllers and templates for different states. In Ionic, the open source and popular router called UI-Router is used for creating states/routes. We will learn about it in detail in the next chapter.

In the `app.js` config block, the states are defined using the `$stateProvider` object. In this template, an abstract state is created for the tabs as it will be shown for all views and then the states for subsequent views are created.

In `index.html`, the `<ion-nav-bar>` directive is used to display a top navbar on all the views. It also contains the `<ion-nav-view>` directive that will act as a placeholder for all the default layout/templates associated with the abstract state for showing the tabs on all views of the app:

```
<ion-nav-bar class="bar-stable">
  <ion-nav-back-button>
  </ion-nav-back-button>
</ion-nav-bar>
<ion-nav-view></ion-nav-view>
```

All the template files are present in the `templates` folder inside the `www` folder. The `tabs.html` file is associated with the main `tabs` state and uses the `<ion-tabs>` directive, which consists of one or more `<ion-tab>` directives to represent each tab. We can set specific properties of the directive such as `icon-on` and `icon-off` to show an active icon and non-active icon respectively. We also create named `<ion-nav-view>` directives under each tab directive to create containers for showing templates for different tabs:

```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">
  <!-- Dashboard Tab -->
  <ion-tab title="Status" icon-off="ion-ios-pulse"
    icon-on="ion-ios-pulse-strong" href="#/tab/dash">
    <ion-nav-view name="tab-dash"></ion-nav-view>
  </ion-tab>
  <!-- Chats Tab -->
  <ion-tab title="Chats" icon-off="ion-ios-chatboxes-outline"
    icon-on="ion-ios-chatboxes" href="#/tab/chats">
    <ion-nav-view name="tab-chats"></ion-nav-view>
  </ion-tab>
  <!-- Account Tab -->
  <ion-tab title="Account" icon-off="ion-ios-gear-outline"
    icon-on="ion-ios-gear" href="#/tab/account">
    <ion-nav-view name="tab-account"></ion-nav-view>
  </ion-tab>
</ion-tabs>
```

The `templates` folder also has a separate file for each tab view named `tab-<view_name>.html`. The code in each file consists of one `<ion-view>` directive on the top with the `view-title` attribute to set the header bar for our app. It also consists of an `<ion-content>` directive to contain the actual content of that particular view.

The controllers for each of the views are bound in the `app.js` in the respective state representing the tab view for account, chats, and dashboard.

## The sidemenu template

The sidemenu template contains a layout where a side menu drawer is used for primary navigation. Ionic Project can be created with this template by mentioning the name of this template in the Ionic start command as follows:

```
$ ionic start MenuDemo sidemenu
```

This template has only the `<ion-nav-view>` directive in the body tag of `index.html`. The main state that is marked as abstract in `app.js` will load the template `templates/menu.html` and associate the controller `AppCtrl` with it. According to the other states defined in `app.js`, respective templates will be loaded into the `menuContent ion-view` directive present in `menu.html`. The code for `menu.html` should be as follows:

```
<ion-side-menus enable-menu-with-back-views="false">
  <ion-side-menu-content>
    <ion-nav-bar class="bar-stable">
      <ion-nav-back-button>
      </ion-nav-back-button>
      <ion-nav-buttons side="left">
        <button class="button button-icon button-clear
          ion-navicon" menu-toggle="left">
        </button>
      </ion-nav-buttons>
    </ion-nav-bar>
    <ion-nav-view name="menuContent"></ion-nav-view>
  </ion-side-menu-content>
  <ion-side-menu side="left">
    <ion-header-bar class="bar-stable">
      <h1 class="title">Left</h1>
    </ion-header-bar>
    <ion-content>
      <ion-list>
        <ion-item menu-close ng-click="login()">
          Login
        </ion-item>
        <ion-item menu-close href="#/app/search">
          Search
        </ion-item>
        <ion-item menu-close href="#/app/browse">
          Browse
        </ion-item>
      </ion-list>
    </ion-content>
  </ion-side-menu>
</ion-side-menus>
```

```
<ion-item menu-close href="#/app/playlists">
    Playlists
</ion-item>
</ion-list>
</ion-content>
</ion-side-menu>
</ion-side-menus>
```

The ion-content directive in this file contains an `<ion-side-menu-content>` directive, which contains the view displayed in the center or the main view. It contains another subview `<ion-nav-view name="menucontent">`, which will show respective templates for each state representing the menu item views. The links to the views are contained in the `<ion-content>` directive under `<ion-list>`.

## The maps template

The maps template contains a layout where a map is loaded on the main view itself. It includes the controller for the view and a directive for showing the map on Ionic App. It has a feature of showing the current position on the map. In order to create a project using this template, please use the following command:

```
$ ionic start MapsDemo maps
```

The `index.html` file contains an `<ion-header>` directive, an `<ion-content>` directive containing the custom `<map>` directive, and the `<ion-footer>` directive. It is important to load the Google Maps API script to display the map. The controller is associated on the `body` tag using the `ng-controller` directive attribute:

```
<body ng-app="starter" ng-controller="MapCtrl">
<ion-header-bar class="bar-stable">
    <h1 class="title">Map</h1>
</ion-header-bar>
<ion-content scroll="false">
    <map on-create="mapCreated(map)"></map>
</ion-content>
<ion-footer-bar class="bar-stable">
    <a ng-click="centerOnMe()" class="button button-icon icon-ion-navigate"></a>
</ion-footer-bar>

<script src="https://maps.googleapis.com/maps/api/js?
key=AIzaSyB16sGmIekuGIvYOfNoW9T44377IU2d2Es&sensor=true">
</script>
</body>
```

## E-commerce sample app – BookStore

From the subsequent sections, we will be creating a sample app while learning about different Ionic components and features. Using this approach, you will be able to develop a complete app at the end of this chapter. The code samples used in the following chapters will be available online.

E-commerce is a hot category for Mobile Apps these days, so we will be creating an e-commerce app for books and will call it BookStore. The features of this app and its basic architecture is discussed here.

## Features

The e-commerce sample app BookStore will be a basic shopping and book management app that has a list of books with the ability to buy or rent them. The proposed features we intend to cover in the code samples are as follows:

- List of book categories
- Book listings under categories and popular/featured listings
- User account section:
  - Login/register
  - Profile
  - Purchased books/rented books
  - Maps
- Shopping section:
  - Add to cart
  - Edit cart
  - Checkout

## Architecture and design

We will be using the side menu template as a skeleton for this app. The side menu will contain links for the user account section and the categories list. We will be using a dummy REST API to source data for the mobile application. The Ionic App will contain a basic service layer consisting of multiple Angular services to integrate with this service.

## **Summary**

We learned to start building an Ionic App using multiple options and different skeleton templates. The chapter also included information about important Ionic directives used for basic content in the Mobile App. We have also discussed the sample app we will be developing during the next chapters.



# 4

## Navigation and Routing in an Ionic App

In any Mobile App, navigation and routing plays the most important part. It defines the backbone of the mobile experience and provides ease of access if implemented perfectly. In this chapter, we will discuss the various options available in Ionic Framework to define routing in an application. Ionic Framework has included an open source routing module called **UI Router**, which was developed along with the Angular UI bootstrap library. UI Router is a preferred routing module over the in-built Angular `ngRoute` module. The following important topics will be discussed in this chapter:

- Introduction to Angular UI Router:
  - States and URLs
  - Nested states and views
  - Multiple and named views
  - State parameters
  - State events and resolve
- Ionic header and footer
- Ionic tabs
- Ionic side menus
- Navigation and back menu
- Navigation and layout to be used in BookStore

## Introduction to Angular UI Router

The core of Ionic Framework is an open source routing module called Angular UI Router. It implements states that are a part of a state machine represented by the complete app. In a normal Angular app, we use `ngRoute`, which defines different routes, each of which can be associated with only a single `ng-view` and one corresponding `templateUrl`. In the UI Router, routes are represented by states (discussed in the following chapter).

## States and URLs

In an app using the UI Router, the views are not tied up to the URL and hence you can change the parts of the app even without changing the URL. In any mobile app, the views are not so simple that they can be changed wholly but there is a complex hierarchy of views and sub-views that change based on different states. Due to this reason it is better to maintain states instead of routes and hence Ionic chose to use the Angular UI Router instead of `ngRoute`. States are also defined in the `config` section of an angular module.

We will learn how to create a state by adding a simple view for our sample app. A new state is created with the name `homeView`, which will be the default view shown after the app is bootstrapped. It will be associated with a controller and a template string or template URL similar to the route:

```
angular.module('BookStore', ['ionic'])
.config(['$stateProvider', '$urlRouterProvider',
  function mainAppConfig($stateProvider, $urlRouterProvider) {
    $urlRouterProvider.otherwise('/home')
    $stateProvider.state('homeView', {
      url: '/home',
      template: '<p>App Home View!</p>',
      controller: 'HomeController'
    });
  }
])
```

The state that we have created will be accessible using the URL `http://<domain-name>/home` and will be opened by default because of the code `$urlRouterProvider.otherwise('/home')`.

We can associate a template like the previous one or give a URL to the `html` template using the `templateUrl` property on the state. Whenever this state is loaded, the template HTML code will be injected into the `<ion-nav-view>` directive of the parent state. In this case, for the root state, we should be adding the `<ion-nav-view>` directive to the `index.html` file with some initial content:

```
<ion-nav-view>
  Loading View...
</ion-nav-view>
```

The perfect way to decide whether a state should exist or not is by ensuring that a state adheres to the following principles:

- A state represents a section of your app that is navigable from one or multiple other states and displays important information about your app
- A state encapsulates important markup and logic in terms of a template and a controller
- Sometimes, multiple states would have some part of the view common and the best approach to implement such a requirement is to create separate states and use state hierarchy to model the commonalities

## Nested states and views

We have already seen how to create a simple state, but we would hardly have an app with simple states. In order to represent a complex view, we would divide our state into some sub states representing sub sections of the views.

We can create nested states in the following multiple ways.

## Using the dot notation

In order to create a nested state, we can use `.` to write the name of the child state after the parent state. If the parent state is named as `homeView`, we can register a nested state `menu` by naming it `.state ('homeView.menu', {})`. The Angular UI Router module by default sets the state before the dot operator as the parent of the state object.

## Using the parent property

Nested states can also be created by using a `parent` property on the state object passed as a second argument to the `state` method of `$stateProvider`. The name of the parent state can be set as a string value to the `parent` property:

```
$stateProvider.state('menu', {  
  parent: 'homeView',  
  templateUrl:'homeView.menu.html',  
  controller: 'MenuController'  
})
```

## Using object-based states

We can use objects instead of strings to set the `parent` property of the state object:

```
var homeView = { name:'homeView', ... };  
var menu = { name: 'homeView.menu',  
            parent: homeView,  
            templateUrl:'homeView.menu.html'  
}
```

States can be registered in any order and a child state can be registered before the parent state. It will queue the child state registration until the parent is registered.

No two state names can be the same. With the dot notation or without dot notation, the names of the states need to be different even if the two states have different parents.

## Views for nested views

If the application is in a particular state, and the state is active, then all its ancestor states are active by default. The child states will load their templates into their parents' `<ion-nav-view>` directives.

## Ways to transition to a state

In order to transition to a specific state or make any state active, there are multiple ways available.

You can use an anchor tag and set the `href` attribute to `#` followed by the URL for the state. For example, if the URL is set to `/home/main`, then the `href` attribute value should be `#/home/main`:

```
<a href="#/home/main"> Main Link </a>
```

Another way to link states in HTML is to use the `ui-sref` directive, which allows the developer to pass in the state name instead of state URL:

```
<a ui-sref="homeView.main"> Main Link </a>
```

If we want to transition to a particular state from JavaScript, then we can use a method on the `$state` object that can be injected into any controller. Please note that in `ui.sref` you give the value starting from the parent template. For example, if you are giving a link in `index.html` then you should give `ui-sref='homeView.main'` but if the link is in `homeView.html` then give `ui-sref='main'`:

```
<a ui-sref="homeView.main"> Main Link </a>
```

We will be working with a new Ionic Project using a blank starter project template and use the code snippets given along with the chapters to build the BookStore e-commerce sample app.

## Abstract state

In the UI router, there is a provision to define an abstract state, which can have child states but it cannot be activated itself. This state cannot be transitioned to and it is activated automatically when one of its descendants is activated. The use cases for using an abstract state is when we want to use a state for layouts only, when a URL needs to be appended to all states, and to provide common data or resolved dependencies to all child states.

Abstract states would still require a `<ion-nav-view>` directive in the template:

```
.state('homeView', {  
  abstract: true,  
  url: '/home',  
  templateUrl: 'homeView.html'  
});
```

## Multiple and named views

In a complex mobile app, we would want multiple sections of our app to change according to different states. In Ionic also, we can have more than one `<ion-nav-view>` in a single template by giving them a named attribute. We can have only one maximum unnamed `ion-nav-view` in a template.

When setting multiple views in a state, we have to use the `views` property, which will contain the name of the views as keys and objects for each view (`templateUrl`, `controller`, and so on) as values.

Abstract states would still require a `<ion-nav-view>` directive in the template:

```
$stateProvider
.state('homeView.books', {
  abstract:true,
  url:'/books',
  templateUrl: 'homeView.books.html'
})
.state('homeView.books.list',{
  views: {
    filters: {
      templateUrl:'homeView.books.filters.html'
    },
    list: {
      templateUrl:'homeView.books.list.html'
    }
  }
});
```

In a state where the `views` property is set, it automatically ignores the template, `templateUrl` or `templateProvider`, so if you want to set a template, then you need to define an abstract state as a parent defining the layout. Also, remember that in Ionic, by default `<ion-nav-view>` has such CSS styling that it takes the whole screen size and the last named view will always be on top. You have to write custom CSS styling to achieve your design.

## **View names – relative versus abstract**

In the previous example, by default the named view is expected on the parent state. If we want to reference a specific named view of any descendant state, then we can use `@` to write the exact view like `<view-name>@<state-name>`.

## **State parameters**

URL routing is very important along with the state mechanics defined for the app. Using the power of URL routing we can send data or parameters in the URL itself, which helps in identifying the state along with some unique identifiers for the state. There are various types of parameter that can be passed to a route or state.

## Basic parameters

The URL can have dynamic parts, which can contain any value, and the controller should be able to access the value passed. You can use a `:` character or `{ }` to define a dynamic URL part, which is called a basic parameter:

```
$stateProvider
  .state('homeView.books.detail', {
    url: "/books/:bookId",
    templateUrl: 'books.detail.html',
    controller: function ($stateParams) {
      // If we got here from a url of /contacts/42
      expect($stateParams).toBe({bookId: "42"});
      // expect() is a method of Jasmine Framework Unit Testing
      // Code
    }
  })
})
```

Alternatively, the basic parameter can also be defined using curly braces:

```
url: "/books/{bookId}"
```

## Regex parameters

We can restrict the values passed in the `url` parameters by using a regular expression inside curly brackets in the `url` value of the state. For example:

```
url: "/books/{bookId:[0-9]{1,8}}"
```

The previous line specifies that the `bookId` parameter can be digits only and its length can vary from 1 to 8.

## Query parameters

Query parameters are similar to the HTML query string parameters commonly used. You can send parameters in the URL itself after the `?` symbol as key value pairs.

### Single parameter

Sample URL structure to represent single query parameter is as follows:

```
url: "/books?bookId" // will match to url of /books?bookId=value
```

## Multiple parameters

Sample URL structure to represent multiple query parameters is as follows:

```
url: "/books?bookId&category" // will match to  
/books?bookId=v1&category=c1
```

## The \$stateParams service

This is an in-built service in the UI Router module, which provides access to each state parameter in the controller. It is important to note that parameters only specific to that state and not its descendants will be available in the controller. An example of URLs and corresponding \$stateParams object is as follows; in this example it is mentioned how the bookId parameter can be accessed using \$stateParams:

```
// If url is of type 'books/:bookId/details?section'  
// Actual Url hit is /books/23/details?section=4  
console.log($stateParams);  
// Console Output:  
{ bookId: 23, section: 4 }
```

## State events and resolve

Important angular events are fired during the management of states in an app. Whenever the state changes from one state to another, there are a handful of events that are broadcasted from the \$rootScope and are available for all the controller scopes to listen. If we want to handle them globally, we can use the \$on method on the \$rootScope variable to handle them. The code would look like this:

```
$rootScope.$on('<event-name>', function handler(eventArgs ...){});
```

The lists of events available are as follows:

Event	Description
\$stateChangeStart	This event is fired when state transition begins: <pre>\$rootScope.\$on('\$stateChangeStart', function(event, toState, toParams, fromState, fromParams) { ... });</pre>
\$stateNotFound	The event is fired when the state requested for is not found in that app. The event is broadcasted allowing any handlers a single chance to deal with the error.
\$stateChangeSuccess	This event is fired when the state change is successful, for example, the transition from one state to another has ended.

Event	Description
\$stateChangeError	This event occurs if there is some error while transitioning from one state to another. This event also occurs if any error is found in the resolve functions. One must listen to this error to catch all errors.

## Resolve

Each and every state can have a property `resolve`, which is helpful to provide content or data to the controller. `resolve` is an optional map of dependencies that should be injected to the controller.

A useful feature of `resolve` is that it can also contain promises, which can help us to get data using AJAX requests. Also, the controller is not initialized for a state before all the `resolve` dependencies. The `resolve` object contains key and value pairs. The key can be any string whereas the value can be a function or a string. If the value of `resolve` is a string, it represents a service in the same module, and if it is a function, it should return a value or any promise.

The code for an example `resolve` block in a state definition looks like this:

```
$stateProvider.state('homeView', {
  url: '/home',
  resolve: {
    information: function(infoService) {
      return infoService.getInfo();
    },
    about: 'aboutService'
  }
});
```

## Ionic header and footer

Ionic has many directives to help common layouts in mobile apps. Ionic headers and footers are important directives for showing proper context for any view in a mobile app. The exact directives, name and their usage are detailed as follows.

## The **<ion-header-bar>** directive

This directive adds a fixed header bar above any content. You can also add a subheader by adding a CSS class `bar-subheader` on the second `<ion-header-bar>` directive, which will be shown below the primary header. The sample code for usage is as follows:

```
<ion-header-bar align-title="left" class="bar-positive">
  <div class="buttons">
    <button class="button" ng-click="back()">Back</button>
  </div>
  <h1 class="title">All about Books</h1>
  <div class="buttons"> <button class="button">Logout</button>
  </div>
</ion-header-bar>
```

There are two attributes available on the directive to add functionality to the header bar as shown in the following table:

Attribute	Type	Details
<code>align-title</code> (optional)	string	This property tells how to align the title. By default it will align as per the platform, for example, left in Android and center in iOS. Available: <code>left</code> , <code>right</code> , or <code>center</code>
<code>no-tap-scroll</code> (optional)	boolean	It will control the property of tapping the header to scroll the view to the top. If set to <code>true</code> , the view will be scrolled and the opposite if set to <code>false</code> . It defaults to <code>false</code> . Available: <code>true</code> or <code>false</code>

## The **<ion-footer-bar>** directive

This directive adds a fixed footer bar below some content similar to the header bar. There can be a subfooter by adding the class `bar-subfooter`. The sample code for usage is as follows:

```
<ion-footer-bar align-title="left" class="bar-assertive">
  <div class="buttons">
    <button class="button">Back</button>
  </div>
  <h1 class="title">All about Books</h1>
  <div class="buttons" ng-click="logout()">
    <button class="button">Logout</button>
  </div>
</ion-footer-bar>
```

There is only one attribute available on the directive to add functionality to the footer bar as shown in the following table:

Attribute	Type	Details
align-title(optional)	string	This property tells how to align the title on the footer. Available: left, right, or center

## Ionic Tabs

Ionic Tabs is a collection of important directives for creating a tab-based layout for a section of your mobile app. We have already seen how the tabs directive is used in the `tabs` starter template.

Ionic Tabs is a collection of two directives and one service. The directives are `<ion-tabs>` used to define the collection or group of tabs, and the `<ion-tab>`, directive, which is used to define a specific tab. The service `$ionicTabsDelegate` is used to control the tabs from JavaScript in order to select a specific tab.

### The `<ion-tabs>` directive

It provides a multi-tabbed interface with a set of tabs and multiple views that can be switched. The tabs CSS classes are used to set the styling of the tabs group. The class name `tabs` should be used along with the following classes for specific styles:

CSS class	Description
<code>tabs-&lt;theme_name&gt;</code>	Defines the color of the tabs, for example, <code>tabs-positive</code> , <code>tabs-calm</code> , and so on.
<code>tabs-icon-only</code>	It will show tabs with icons only. You have to include the icon in the <code>tab</code> directive.
<code>tabs-icon-top</code>	This class puts the icon at the top and text at the bottom.
<code>tabs-icon-left</code>	This class puts the icon on the left.

The `<ion-tabs>` directive has one attribute that enables you to pass a custom handle for the delegate as shown in the following table:

Attribute	Type	Details
delegate-handle (optional)	string	The handle used to identify these tabs with <code>\$ionicTabsDelegate</code> .

## The `<ion-tab>` directive

Inside the `<ion-tab>` directive, we can use one or multiple `<ion-tab>` directives. The tab content can be included in the `<ion-tab>` directive, which will be shown only when the specific tab is selected. Each `<ion-tab>` has its own navigation history and should include its own named `<ion-nav-view>`.

The following is the usage of this directive:

```
<ion-tab
  title="Custom Tab"
  icon="ion-star"
  href="#/book/favourites"
  on-select="goToFavourites()"
  on-deselect="unsetFavourites()">
</ion-tab>
```

The important attributes available on the `<ion-tab>` directive and their functions is outlined in the following table:

Attribute	Type	Details
title	string	The title of the tab view
href (optional)	string	The link that will be opened if the tab is tapped
icon (optional)	string	The icon of the tab, which if set, is default for <code>icon-on</code> and <code>icon-off</code>
icon-off (optional)	string	The icon of the tab when it is not active
icon-on (optional)	string	The icon of the tab when it is active
badge (optional)	expression	A number or small character shown on the icon tab
on-select (optional)	string	This method will be called when tab is selected

Attribute	Type	Details
on-deselect (optional)	string	This method will be called when tab is deselected

## Ionic side menu

Ionic side menus also consist of multiple directives that help in achieving a layout with one or many side menus. It defines the layout of a section of your app. All the directives for this purpose are defined in this section.

### The <ion-side-menus> directive

It is container for the `ionSideMenu` directive and the main content. A left-side menu, right-side menu, or both can be added to the layout of your app. This directive/element must have at least two children, one should be an `<ion-side-menu-content>` directive, which will load the views of the app, and one or more `<ion-side-menu>`.

The different attributes available to be used with this directive are as follows:

Attribute	Type	Details
enable-menu-with-back-views (optional)	bool	This attribute determines whether the side menu is enabled when the back button is showing. If it is set to <code>true</code> then the <code>menu-toggle</code> elements will be shown, and if the value is <code>false</code> , they will be hidden.
delegate-handle (optional)	string	The handle used to pass the delegate to manage side menus.

### The <ion-side-menu> directive

This directive is the place for side menu content. The side menu list view and other content can be included in this directive.

The following is the usage of this directive:

```
<ion-side-menu
  side="left"
  width="200"
  is-enabled="shouldLeftSideMenuBeEnabled() ">
</ion-side-menu>
```

This directive has only three attributes: `side` for defining the side it will slide from, `width` to set the width of the side menu container, and `is-enabled` to set an expression/function to evaluate whether the specific side menu should be enabled or not.

## The `<ion-side-menu-content>` directive

This is the container for the main visible content. It can contain `<ion-nav-view>` to show different views based on the state changes. It is always a sibling to the `ionSideMenu` directive.

The following is the usage of this directive:

```
<ion-side-menu-content  
  edge-drag-threshold="true"  
  drag-content="true">  
</ion-side-menu-content>
```

This directive has two attributes—`edge-drag-threshold`, which defines whether content drag can be enabled or start below a threshold distance from the edge, and `drag-content`, which enables whether content inside this directive can be dragged or not.

## Other important directives

In order to work with side menus there are some more directives to be used:

- `menu-close` is an attribute directive that closes a currently opened side menu. This can be used on a link to navigate to a new view, which would close the side menu. It also resets the view's navigation history and removes the back button.
- `menu-toggle` is an attribute that can be set on a button or icon used to toggle a side menu. The specific side menu such as `left` or `right`, can be set to this attribute value.
- `expose-aside-when` is an attribute that controls when the side menu should be kept open. In the case of large tablets, the side menu will always be opened. This directive should be given in the `ionSideMenu` directive.

## Navigation and back menus

Ionic Framework has strong navigation support in the form of various directives such as `ionNavView`, `ionView`, `ionNavBar`, and so on. Ionic stores and maintains the navigation history in the mobile app. It dictates the transitions appropriately. An app section can have multiple views, which maintain separate navigation histories.

The UI Router module and states are an integral part of the design for the navigation concepts. The `<ion-nav-view>` directive has already been explained in the first section while defining the UI Router module.

`<ion-view>` is another directive that should be used as a container for the content of any view. It should have a `<ion-content>` directive and can have header and navbar information. It defines the title of the parent `ionNavBar` using the `view-title` attribute. There are other attributes such as `cache-view` to enable caching, `hide-back-button` to define whether the back button should be displayed or not, and `hide-nav-bar` attribute to define whether to hide the parent `ionNavBar` or not:

- `<ion-nav-bar>` is another directive that is created along with the `<ion-nav-view>` directive and shows a top bar that will be modified as soon as the state changes. We can add a back button to be shown using a `<ion-nav-back-button>` directive. We can also add some custom buttons using the `<ion-nav-buttons>` directive.
- `<ion-nav-title>` is a directive that can be used inside the `<ion-nav-view>` directive to override the title set in the navbar with any custom title or HTML content as a title.

## Navigation and layout to be used in BookStore

In our sample app, BookStore, we will use a side menu layout throughout. The side menu will contain the links to different sections such as account, orders, checkout, and categories.

The top categories will also be directly accessible from the side menu. The side menu content will show the main views. All the views will have navbar buttons to show **Logout** and **User Account** icon buttons for swift navigation.

## **Summary**

We have learned all about different ways to implement navigation and routing in a Hybrid Mobile Application. We have also learned how to define navigation and view hierarchy for complex views and Mobile Apps. We have also learned about all the available directives and options to use different functionalities to define views and navigation in a Mobile App. In the next chapter, we will come to know about different available components that can be used in our sample e-commerce app.

# 5

## Accessorizing Your App with Ionic Components

We have learnt to build the skeleton and base for a Mobile App. An actual app would contain multiple use cases and complex views, which would include many more elements and components. Ionic Framework equips developers with a multitude of reusable components and directives that they can use to develop apps rapidly. There are two categories of reusable components: CSS components for styling elements, and JS components for logic and interactions. The topics we will be discussing in this chapter are:

- Ionic CSS components
  - Header and footer
  - Buttons
  - Lists
  - Cards
  - Forms
  - Input elements
  - Tabs
  - Grid
  - Utility styles
- Ionic JS components
  - Actionsheet
  - Backdrop
  - Form inputs

- Gestures and events
- Lists
- Loading
- Modal
- Popover and popup
- Spinner
- Miscellaneous components

We will learn about the usage and examples of each of the mentioned components.

## Ionic CSS components

Mobile Apps are required to provide an engaging user experience. The styles and look and feel are represented by CSS in a Hybrid Mobile App. It takes a lot of time for a developer to write the CSS styles for multiple device screen sizes and so many UI elements. Ionic Framework provides reusable CSS styles that help developers in creating awesome Mobile Apps and decrease the development time rapidly. We will be discussing different CSS styles available for developers to use and customize.

## Header

The header corresponds to the static and fixed region on the top of the screen, which can contain a title in the center and buttons on the left or right. Headers have their own CSS class and also for showing different header colors. For example:

```
<div class='bar bar-header bar-{color}'>
  <h1 class='title'> Header </h1>
</div>
```

We can replace {color} with the color themes available in Ionic, for example, light, stable, positive, calm, balanced, energized, assertive, royal, or dark. There is another class to display a sub header too, along with the header:

```
<div class='bar bar-subheader'>
  <h2 class='title'>Sub Header</h2>
</div>
```

We should remember to use the CSS class that has subheader to the `<ion-content>` directive.

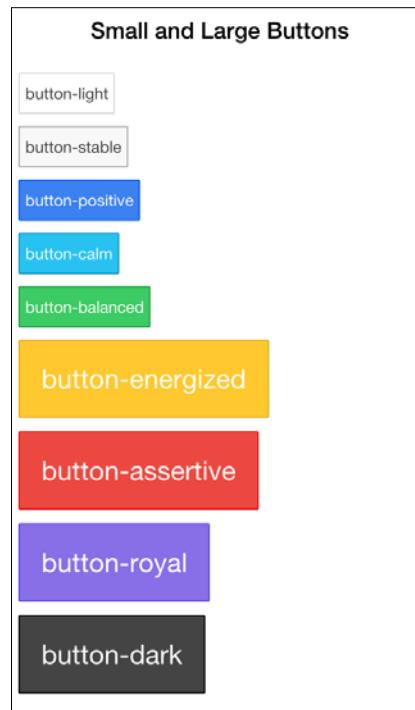
## Footer

Footer classes are exactly similar to headers in that they have the class `bar-footer` for the footer and `bar-{color}` representing a specific color for the footer. If we want to place buttons in the header or footer, we can place them before or after the title in HTML code according to where we want them to appear:

```
<div class="bar bar-footer">
  <button class="button">Home</button>
  <div class="title">Title</div>
  <button class="button">Login</button>
</div>
```

## Buttons

Buttons are an important part of the UI as they drive actions in the whole Mobile App's UI. Ionic provides a variety of CSS styling options to display different buttons. Ionic has the class `button` for styling any button and has all the default colors that are available to headers/footers. The CSS class for any color would be `button-{color}` where color can be replaced by `balanced`, `assertive`, `calm`, and so on. The following image shows different sized buttons using all color themes:



Buttons can be of different sizes too. There are two classes, `button-large` and `button-small`, that provide two variances for having a larger button than normal and a smaller button than normal, respectively. If we want to create full-width buttons for Mobile Apps, there are two CSS classes. The `button-block` applies the `display: block` CSS property and allows the button to take 100% of its parent width. If we add the `button-full` CSS class to a button, then the button would stretch across the entire width and would not have left or right borders.

If we want to show buttons with an outline only, then we can use the CSS class `button-outline`. The background color is transparent for this CSS class. Also, the outline color and text color are the same as the color class used on the button.

In order to show a button without a background or a border, the `button-clear` CSS class can be used.

## Icon buttons

Icons can be added to any buttons by using built-in icons called **Ionicons** (<http://ionicons.com>) or any custom font pack. Icons can also be set as a child element to the `button` element:

```
<button class="button">
  <i class="icon ion-home"></i>
</button>
<button class="button icon-left ion-settings">
  Text
</button>
```

We can decide the alignment of the icon by using the classes `icon-left` or `icon-right`. Icons can be put directly on a button or even on a link tag given `button` class and `icon` class along with it. The same buttons can be used in the header or footer too.

## Button bar

If we want to group multiple buttons together, we can use this CSS class for creating a bar of buttons. The CSS class to be used is `button-bar` and the usage is as follows:

```
<div class="button-bar">
  <a class="button button-balanced icon-left ion-home">Home</a>
  <a class="button button-calm icon-left ion-
    settings">Settings</a>
```

---

```
<a class="button button-assertive icon-left ion-chevron-
    right">Next</a>
</div>
```

A button bar can be added to the header or footer also.

## Lists

Mobile views due to limited space cannot show big tables and grids. List is the most popular and widely used UI design pattern used for mobile views. The list contains a collection of list items, which can contain any content ranging from text, images, thumbnails, icons, and so on:

```
<ul class="list">
  <li class="item">
    </li>
  </ul>
```

In Ionic Framework, there are CSS classes as well as directives for multi-featured lists. There are different classes to support dividers, icons, buttons, thumbnails, and so on.

## List dividers

Apart from the list items, there can be dividers added to the list. There is a class, `item-divider`, that will appear to be dividing multiple list items:

```
<div class="list">
  <div class="item item-divider">
    Display Settings
  </div>
  <a class="item" href="#">
    Icons Display
  </a>
</div>
```

## List icons

Generally, list items have mixed content and icons play an important role in highlighting any list item. The icons can be placed on either the left or right side. Icons can be chosen from the built-in Ionicons or any custom font pack chosen as a font. The class `item-icon-left` or `item-icon-right` has to be included for every item. If icons are required on both sides, both CSS classes can be used.

## List buttons

A `<button>` element can also be placed on the left or right by adding the classes `item-button-left` and `item-button-right` respectively.

## Item avatars or thumbnails

In order to display images on list items, the `item-avatar` or `item-thumbnail` classes can be used. An avatar is a smaller image that is rounded, and a thumbnail is a bigger picture generally in a square shape. The following code snippet shows examples for all the list item types:

```
<div class="list">
  <a class="item item-icon-left item-icon-right" href="#">
    <i class="icon ion-chatbubble-working"></i>
    Call Support
    <i class="icon ion-ios-telephone-outline"></i>
  </a>
  <a class="item item-avatar" href="#">
    
    <h2>Demo User</h2>
    <p>Bought 23 books since 2014.</p>
  </a>
  <a class="item item-thumbnail-left" href="#">
    
    <h2>Bestseller Book</h2>
    <p>New Book</p>
  </a>
</div>
```

## Cards

Cards is a new UI design pattern that has become popular on mobile screens as it makes it easy to segregate important information and content. Ionic provides special CSS classes to create strong mobile views. Cards create a box around the content and add a box shadow to it, as follows:

```
<div class="card">
  <div class="item item-text-wrap">
    Card dummy text content.
  </div>
</div>
```

Cards can also contain list items and the `list` class can be used along with the `card` class. Item dividers can be used in the cards to create distinguished content.

## Forms

In order to display forms in Ionic, lists can be used to display a set of input controls or input items. Both the classes `item` and `item-input` should be used to display a user form element.

By default, the input elements take 100% width. For an input text element, you can add the classes on the `label` tag and use the `placeholder` property to simulate the input's label. When the user types some text, the placeholder gets hidden and is overridden by user input:

```
<label class="item item-input">
  <input type="text" placeholder="Dummy Text">
</label>
```

If you need to put a `label` and `input` tag in the same row, then put a new `span` element with the class `input-label` displaying the label on the left column of the item row:

```
<label class="item item-input">
  <span class="input-label">Name</span>
  <input type="text">
</label>
```

In order to stack labels on top of the `input` elements, use the class `item-floating-label` on the `item` row containing the `label` and the `input` element. Ionic has a special style for floating labels, which animate from being placeholders to stacked labels when the user starts typing in the `input` element.

As mentioned, the `input` elements take 100% width but if you want to have inset form elements then you can encapsulate a form list in a card or either add a class `list-inset` to the `list` element. If you want to inset only a single `input` element and not the entire list then use the class `item-input-inset` on the `item` element.

## Input elements

Apart from standard `input` element tags there are other `input` elements such as `select` box, `checkbox`, `radio`, and so on, which can be used in forms on mobile views. Ionic provides various CSS styles for these input elements, which will be discussed here.

## Checkbox

A checkbox in Ionic is similar to the HTML counterpart with a good and elegant UI. The class `item-checkbox` needs to be added to the container for the checkbox, and its label, the second class `checkbox`, is added to the label element:

```
<ul class="list">
  <li class="item item-checkbox">
    <label class="checkbox">
      <input type="checkbox">
    </label>
    Checkbox Option1
  </li>
</ul>
```

## Radio button list

Similar to checkbox, if only a single option has to be selected out of a list, a radio button list control can be used. It will act in the same way as HTML `radio` input elements.



The name for all the `input-type radio` elements should be the same.



The icon that will be shown on the right of the selected radio element can be defined in the HTML itself. Three CSS classes will be used for the radio list element, `item-radio` for the container, `item-content` for the label text to be shown, and `radio-icon` for the icon for the selected item:

```
<div class="list">
  <label class="item item-radio">
    <input type="radio" name="group">
    <div class="item-content">
      Radio Option 1
    </div>
    <i class="radio-icon ion-checkmark"></i>
  </label>
</div>
```

## Toggle

A toggle control is a mobile-specific component used in lieu of a checkbox. It is more intuitive for mobile devices and provides a rich user experience. The code for the `toggle` elements requires a `checkbox` element wrapped in a `label` with the class `toggle`:

```
<div class="list">
  <li class="item item-toggle">
    <label class="toggle">
      <input type="checkbox">
      <div class="track">
        <div class="handle"></div>
      </div>
    </label>
  </li>
</div>
```

## Range

The range control in Ionic is a slider to select a specific value. The `range` component can be applied to different color themes according to the Ionic themes:

```
<div class="item range">
  <i class="icon ion-volume-low"></i>
  <input type="range" name="volume">
  <i class="icon ion-volume-high"></i>
</div>
```

The range item will have a left-side icon, right-side icon, and an `input` element of type `range` in between.

## Tabs

We have discussed the `tabs` component and directive in the previous chapters. We can also create tabs in other views by using CSS classes only. It is a horizontal bar present on the bottom of the view containing one or more tab items. The classes `tabs` are used on the container and the class `tab-item` for each item. Tabs can be given a color theme using the CSS classes such as `tabs-calm`, `tabs-assertive`, and so on:

```
<div class="tabs">
  <a class="tab-item">
    Tab 1
  </a>
```

```
<a class="tab-item">  
  Tab 2  
</a>  
<a class="tab-item">  
  Tab 3  
</a>  
</div>
```

Tabs can also use some extra CSS classes to display icons and control their alignment. The list of those CSS classes has been discussed in *Chapter 4, Navigation and Routing in an Ionic App*, in the *Ionic Tabs* section.

## Grid

In developing a UI, creating the layout is like laying the foundation, and a strong grid system helps create solid layouts. In Ionic Framework there is a CSS grid system, which is based on the CSS flexbox layout. It helps in automatically adjusting dynamic columns in a row or rows.

In Ionic we will use the `row` class to represent a row and the `col` class for a column. We can add as many as columns in a row ranging from 1 to n and they will adjust equally. By default, when we use the `col` class, equally spaced columns are created.

We can also explicitly define the size by using classes such as `col-50`, `col-33`, `col-75`, and so on, and the rest of the columns will adjust accordingly. We can also give offset to the columns by adding the keyword `offset` in the column `class` name, for example, `col-offset-50`, `col-offset`, and so on.

In order to align columns vertically among a row, we can use the CSS classes `col-top`, `col-center`, and `col-bottom` respectively. The columns can be made to stack in a specific screen size and become responsive by adding the CSS classes `responsive-sm`, `responsive-md`, or `responsive-lg` for a small screen size (smaller than landscape phone), medium (smaller than portrait tablet), or large (smaller than landscape tablet).

The example code for using a grid or column is:

```
<div class="row">  
  <div class="col col-50">.col.col-50</div>  
  <div class="col">.col</div>  
  <div class="col">.col</div>  
</div>
```

```
<div class="row">
  <div class="col col-75">.col.col-75</div>
  <div class="col">.col</div>
</div>

<div class="row">
  <div class="col">.col</div>
  <div class="col col-75">.col.col-75</div>
</div>

<div class="row">
  <div class="col">.col</div>
  <div class="col">.col</div>
</div>
```

## Utility

There are some utility classes that can be used along with multiple components. Ionic Framework has standard colors that can be used throughout the Mobile App with other styles. Utility color styles can be customized and existing themes can be modified. You can modify color variables in the `_variables.scss` file.

Ionic also comes with an icon font set that contains the most common fonts. These can be used by applying the `icon` class name and the specific class for different icons. For example:

```
<i class="icon ion-email"></i>
```

Also, there are padding classes available to add a padding of 10px between the outer box and the inner content. The class `padding` adds padding to all sides of the element whereas other CSS classes can be used for giving padding on special sides.

For example, `padding-vertical`, `padding-horizontal` for top-bottom and left-right respectively and `padding-top`, `padding-right`, and `padding-bottom`, for respective sides.

## Ionic JS components

CSS is used for styling mobile components but JavaScript is used to write logic and create the user experience in a Hybrid App. Ionic has reusable Angular directives that help developers create smooth mobile-specific experiences. Apart from directives, Ionic also has controllers and services in its library to help create mobile-specific components. We will be discussing the most important components in this section of the chapter.

## ActionSheet – \$ionicActionSheet

The actionsheet component is a sliding panel that comes from the bottom and can contain multiple buttons intended to perform some actions in the Mobile App. This component has been inspired from the actionsheet component in iOS. \$ionicActionSheet is a service in Ionic Library that can be injected in any controller and initiated using the show method.

The show method inputs a set of options mentioned in the following table to control the actionsheet component.



A new isolated scope is created when the show method is called.



Name	Type	Description
buttons	[Object]	If extra buttons need to be shown, passed as array of objects. Each object can contain a text property.
titleText	String	Text for the title of Actionsheet.
cancelText	String	Text shown for a cancel button.
destructiveText	String	Text shown for a destructive button.
buttonClicked	Function	Callback called when any extra button is clicked.
cancel	Function	Callback called on click of cancel button.
destructiveButtonClicked	Function	Callback function for a destructive button.
cancelOnStateChange	Boolean	Value to define whether to cancel actionsheet on state change.
cssClass	String	Custom CSS class applied on actionsheet.

The code for an example is as follows:

```
// Action Sheet Initialization using Properties
var sheetId = $ionicActionSheet.show({
  titleText: 'Manage Books',
  buttons: [ { text: '<b>Add New Book </b>' } ],
  cancelText: 'Cancel',
```

```
destructiveText: 'Delete Book',
cancel: function() {
    // code when cancel button is clicked
},
buttonClicked: function(index) {
    // code which is to be executed on button click
}
});
```

## Backdrop - \$ionicBackdrop

A backdrop is a transparent blackish UI screen that appears behind popups, loading, and other overlays. Multiple UI components require a backdrop, but only one backdrop appears in DOM. In order to use a backdrop, any component can call the `retain` method, and whenever its usage is over the component can call the `release` method.

The code for demonstrating its usage is as follows:

```
$stateProvider.state('homeView', {
    //Show a backdrop for one second
    $scope.action = function() {
        $ionicBackdrop.retain();
        $timeout(function() {
            $ionicBackdrop.release();
        }, 1000);
    };
});
```

## Form inputs

Apart from the normal input elements to take text format inputs, there are special form input directives for checkboxes, radio, and toggle. We have already discussed the CSS styles used for the same, but you can use directives themselves to achieve the functionality along with the style.

### The <ion-checkbox> directive

This directive adds the required styles to a checkbox. It renders like a normal angular checkbox:

```
ion-checkbox ng-model="isChecked">Checkbox Label</ion-checkbox>
```

## The `<ion-radio>` directive

The `<ion-radio>` directive is also used to style the radio input according to the mobile UI. In order to create a set of radio buttons, please give the same `ng-model` property to all and use `ng-value` to assign a value for a specific radio element:

```
<ion-radio ng-model="option" ng-value="'1'">Option 1</ion-radio>
<ion-radio ng-model="option" ng-value="'2'">Option 2</ion-radio>
```

## The `<ion-toggle>` directive

This directive creates a toggle switch that can manipulate a boolean model:

```
<ion-toggle ng-model="isOpen" toggle-class='toggle-calm'>Open
Drawer</ion-checkbox>
```

## Gestures and events

In a Mobile App, gestures and events are very important in shaping the user experience. The event handling should be perfect and there should be a multitude of events available to developers for adding appropriate actions for different events. In Ionic Framework, there are different event directives that can be used to bind callbacks for those events. Also, there is an `$ionicGesture` service that can be used to programmatically bind the events with callbacks.

## The `$ionicGesture` service

The methods available under the `$ionicGesture` service are given as follows.

### The `on` method

It adds an event listener for a gesture on an element. The signature for the method is as follows:

```
on(eventType,callback,$element,options)
```

The following table lists the details of each argument:

Param	Type	Details
eventType	string	The gesture event to be registered
callback	function(e)	Event listener method to be called when event is triggered

Param	Type	Details
\$element	element	Element on which the event is bound
options	object	Used to set extra options

The method returns an `ionic.Gesture` object, which is used to detach the event later.

## The off method

It removes an event listener for a gesture on an element. The signature for the method is as follows:

```
off(gesture, eventType, callback)
```

The following table lists the details of each arguments:

Param	Type	Details
gesture	ionic.Gesture	The gesture that is to be removed
eventType	string	The gesture name that is to be removed
callback	function(e)	Callback that needs to be deregistered

## Gesture events

A list of gesture events available in Ionic Framework are given in the following table:

Event name	Description
on-hold	It occurs when the touch stays for longer than 500 ms. Long touch events.
on-tap	It occurs when a quick touch is done on an element.
on-double-tap	It occurs on a double tap on a location.
on-touch	It is called immediately when the user starts a touch.
on-release	It is called when the user ends a touch.
on-drag	It is called when one touch is moved around the page. If dragging is allowed, scrolling should be disabled.
on-drag-up	It is called when element is dragged up.
on-drag-right	It is called when an element is dragged right.

Event name	Description
on-drag-left	It is called when an element is dragged left.
on-drag-down	It is called when an element is dragged down.
on-swipe	It is called when a touch moves at high speed in any direction.
on-swipe-up	It is called when a touch moves at high speed moving up.
on-swipe-right	It is called when a touch moves at high speed moving right.
on-swipe-left	It is called when a touch moves at high speed moving left.
on-swipe-down	It is called when a touch moves at high speed moving down.

## Lists

List is the most popular component used in any Mobile App. Generally, all the data displayed in a mobile view is in the form of a list. A list can have multiple list items representing rows. A list item can contain any content ranging from text to icons and custom elements.

### The `<ion-list>` directive

This directive represents a list in Ionic Framework. This directive supports complex interactions such as drag to reorder, swipe to edit, and removing items. The `<ion-list>` directive can consist of one or more `<ion-item>` directives representing each row. There are other directives augmenting its functionality such as `<ion-option-button>` used for swipe to edit buttons, `<ion-reorder-button>` shown while reorder is enabled, and `<ion-delete-button>` shown while deleting a row item.

The attributes available for this directive are given in the following table:

Attribute (all optional)	Type	Details
delegate-handle	string	The handle used for passing the reference of the list with the object \$ionicListDelegate
type	string	Type of list used (options: list-inset or card)
show-delete	boolean	Whether to show delete buttons or hide them
show-reorder	boolean	Whether to show reordering buttons or hide them
can-swipe	boolean	Whether to allow swiping of the list to show option buttons

The following code explains the usage :

```
// Example with complex Scenario
<ion-list ng-controller="AppCtrl"
  show-delete="shouldShowDelete"
  show-reorder="shouldShowReorder"
  can-swipe="listCanSwipe">
  <ion-item ng-repeat="row in rows" class="item-thumbnail-left">
    <h2>{{item.title}}</h2>
    <p>{{item.description}}</p>
    <ion-option-button class="button-positive" ng-
      click="share(item)">
      Share
    </ion-option-button>
    <ion-option-button class="button-info" ng-click="edit(item)">
      Edit
    </ion-option-button>
    <ion-delete-button class="ion-minus-circled" ng-
      click="items.splice($index, 1)">
    </ion-delete-button>
    <ion-reorder-button class="ion-navicon" on-
      reorder="reorderItem(item, $fromIndex, $toIndex)">
    </ion-reorder-button>
  </ion-item>
</ion-list>
```

## Loading – \$ionicLoading

\$ionicLoading is an angular service that controls the display of an overlay representing a loader. The loading indicator can be configured by passing certain options to the `show` method of this service.

The following is the usage of this service:

```
// Sample Module
angular.module('MyApp', ['ionic'])
controller('MyCtrl', function($scope, $ionicLoading) {
  $scope.show = function() {
    $ionicLoading.show({
      template: 'Loading...'
  });
  $scope.hide = function() {
    $ionicLoading.hide();
  };
});
```

The important options available to be passed to the `show` method are:

Field	Type	Details
<code>template</code>	string	HTML content of the indicator
<code>templateUrl</code>	string	Template URL for the HTML content of the indicator
<code>scope</code>	object	The scope to be a child of
<code>noBackdrop</code>	boolean	Whether to hide the translucent backdrop
<code>hideOnStateChange</code>	boolean	Whether to hide the loader during state change
<code>delay</code>	number	Delay (in ms) to show the indicator
<code>duration</code>	number	Time (in ms) after which the indicator will be hidden

## Modal – \$ionicModal

This is a service that controls displaying a popup over the existing UI temporarily. It uses the `<ion-modal-view>` directive in its template to represent the modal DOM element. It is used for generally editing an item or showing a dialog box or a choice box. There are two methods available on the `$ionicModal` service, `fromTemplate` and `fromTemplateUrl`, which takes input as `template string` or `template url` respectively as a first parameter. The second argument to both methods is the `options` object, which is passed to the initialize method of the `IonicModal` controller instance.

## The IonicModal controller

This is a controller instantiated by the `$ionicModal` service. The `ionicModal` method `remove` should be called after the use of the modal is over in order to avoid memory leaks.

The different methods available to this controller are as follows.

### initialize(options)

It creates a new modal controller instance. It takes as an argument an `options` object, which should contain the following properties:

- `scope (object)`: The scope to be a child of.
- `animation (string)`: The animation to show and hide. The default is `slide-in-up`.
- `focusFirstInput (boolean)`: The control whether to focus first input in modal or not.
- `backdropClickToClose (boolean)`: Whether to close modal on clicking backdrop.
- `hardwareBackButtonClose (boolean)`: The control to decide whether modal should be closed using hardware back button or not.
- `show ()`: This method is used to show the modal instance.
- `hide ()`: This method is used to hide the modal instance.
- `remove ()`: This method is used to remove the modal instance from memory.
- `isShown ()`: This method returns a boolean representing whether the modal is shown or not.

## **Popover – \$ionicPopover**

A Popover component is also modeled similarly to modal and has two objects, the `$ionicPopover` service and the `ionicPopover` controller. Popover is a view that floats above the app's content. Popover can be used to display extra information or take input for a choice. The content of a Popover needs to be put inside an `<ion-popover-view>` element.

The `$ionicPopover` service exposes similar methods such as the `$ionicModal` service—`fromTemplate` and `fromTemplateUrl`. The `ionicPopover` controller also has similar events to the `ionicModal` controller. The `initialize` method for this controller has the `options` object as input but it does not contain the `property animation`.

## **Spinner – ion-spinner**

In a mobile UI or views, only a particular section of view needs to be refreshed or loaded and a spinner/loader needs to be displayed in that section. `<ion-spinner>` is a directive that displays loader icons. There are multiple available icons such as spiral, ripple, dots, lines, and so on. The theme colors for Ionic can also be used along with using class names such as `spinner-<theme>`.

The following is the usage of this directive:

```
<ion-spinner icon="spiral spinner-calm"></ion-spinner>
```

## **Miscellaneous components**

There are some extra utility and miscellaneous components available in Ionic Framework. The components can be in the form of directives or services.

## **\$ionicPosition**

It is a utility service used to retrieve the position of DOM elements. This can be used to position elements absolutely over the screen. It has two methods available, `position` and `offset`, which fetch the position relative to its parent or relative to the document respectively. The methods return an object containing the properties `top`, `left`, `width`, and `height`.

## \$ionicConfigProvider

Ionic has a lot of default configurations and styles for different platforms. For example, in iOS, the tab bar is shown below by default, and in Android, the tab bar is shown on top. The Ionic platform exposes a provider to manage these configurations, which is called `$ionicConfigProvider`. This provider can be used during the config phase of your Angular app. The configuration is, by default, set for the specific platform.

The following is the usage of this variable:

```
var myApp = angular.module('MyApp', ['ionic']);
myApp.config(function($ionicConfigProvider) {
  $ionicConfigProvider.views.maxCache(5);
  $ionicConfigProvider.backButton.text('Go Back').icon('ion-chevron-left');
});
```

A list of different configurations that can be set using this provider are as follows:

- `views.transition(transition)`: Animation style while viewing transitions
- `scrolling.jsScrolling(booleanValue)`: Setting usage for native JS scrolling or not
- `backButton.icon(value)`: Setting the back button icon
- `backButton.text(value)`: Setting the text for back button
- `tabs.position(value)`: Setting the position of tabs out of top or bottom
- `navBar.alignTitle(value)`: Setting the alignment side for the navbar's title
- `spinner.icon(value)`: Setting the default icon for the spinner

## Summary

In this chapter we have seen most of the components available in Ionic Framework as directives and services to build excellent Hybrid Mobile Apps. These components cover most of the use cases required for developing Mobile Apps. We have learnt about different CSS styles that can be used to create an engaging user experience in your Ionic Apps. Ionic Library also has JS components that we have discussed in detail in this chapter. We can also mix and match these components to build more complex directives. In the next chapter we will learn how we can integrate our apps with any web service or backend service providers.



# 6

## Integrating App with Backend Services

Mobile Apps are incomplete without data, and the data on the mobile is not enough. We have learnt to start Mobile App projects and create complex views with different components. In this chapter we will learn how to integrate our Ionic Apps with web services to fetch and submit data. Use cases involving data exchange for Mobile Apps include central user authentication/authorization, saving your personalization data, storing images, searching public datasets, storing transactions, and so on. Mobile Apps have become a major source for collaboration and require strong integration to robust backend services to support communication and real-time messaging.

Here, we will first learn about the low-level constructs available in Angular/Ionic to integrate into any web service conforming to JSON/REST standards. There is also another object available that maps to REST entities directly, helping to shorten the integration development effort and time. The topics that will be discussed in this chapter are as follows:

- \$http services
- Ionic services versus factories
- \$resource and REST API
- Demystifying mBaaS
- Integrating with Parse
- Integrating with Firebase

Mobile Apps become meaningful with data, and to empower our Ionic Apps with data we need to have strong backend servers or APIs. We will discuss some cloud-based backends as a service solution, which are super easy to use with our Ionic Apps with minimal effort.

## \$http services

In web technologies, the best way to interact with any web service is through Ajax requests. As Ionic Framework is a Hybrid Mobile framework based on web technologies, it utilizes the power of Ajax to wire up Ionic Apps with any web services.

\$http is an in-built Angular service that is used as an abstraction for native JavaScript Ajax calls. The \$http service has some high-level methods exposed to make HTTP requests using different HTTP methods such as POST, GET, PUT, and so on.

There are different signatures for different methods, but the response for all the methods is exactly the same. All the methods in the \$http service are based on the promise objects, which help in registering success and error callbacks that receive data at a later point in time as these requests are asynchronous in nature.

The two most important methods are .get and .post whose usage is as follows:

```
$http.get('/api/url/resource')
  .then(function(response) {
    // this is success callback
  },function(errorResponse) {
    // this is error callback
  });
}
```

The GET request method takes only one argument, the URL for the request, and registers two callbacks with the method of the promise object returned:

```
$http.post('/api/url/resource',{data:'custom data'})
  .then(function(response) {
    // this is success callback
  },function(errorResponse) {
    // this is error callback
  });
}
```

The POST request method takes two arguments, the URL for the request and the data to be sent in the body of the post request.

The other methods available that have a signature exactly like the GET method are as follows:

- \$http.put
- \$http.head
- \$http.delete
- \$http.jsonp
- \$http.patch

There are default headers sent along with each HTTP request. Some web services require us to send custom headers for authorization or content-type for which the `$http` service exposes a provider to set custom default headers. We can use the provider to set up default headers in the config phase of the Ionic App. We can also use the `$http` object to set default headers in the run block or any controller/service:

```
module.config(function($http) {
  $http.defaults.headers.common['Content-Type'] = 'application/json';
});
module.run(function($http) {
  $http.defaults.headers.post['Authorization'] = 'Basic sdkJKHSmd'
});
```

## The response object

The `response` object passed as an argument to the callback functions contains the following properties:

Name	Type	Description
<code>data</code>	String/ Object	Response body received in the Ajax request, generally JSON is parsed.
<code>status</code>	Number	HTTP status code returned by the request.
<code>headers</code>	Function	Header getter function.
<code>config</code>	Object	The configuration object used to generate the request.
<code>statusText</code>	String	HTTP status text for the response.

The `response` object can be used to extract the data sent by the server and process it.

## The `$http` constructor method

Another way to make an Ajax request is by using the `$http` constructor method and passing the request configuration object directly. For example:

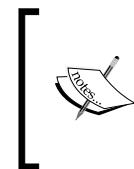
```
var req = {
  method: 'POST',
  url: '/api/url/resource',
  headers: {
    'Content-Type': 'plain/text'
```

```
},
  data: { objectName: 'Value' },
  cache: true
}
$http(req).then(function() {...}, function() {...});
```

The `$http` service also supports caching by passing the `cache` property to the configuration object and setting it to `true`. Angular stores the response temporarily to the `$cacheFactory` object and serves the same request with a response from there.

## Ionic services vs factories

Services/factories in Ionic are used to abstract common business logic and integration to the backend services. A data access layer and business logic layer can be implemented using Ionic service or factory. Ionic service or factory can be used interchangeably and are available as a method on the `angular.module` object.



Generally, a service is used to represent the data access layer whereas a factory is used to implement the entity layer or business logic layer. Both objects are singleton in nature, meaning only once instance is created throughout the app's life cycle, which is shared between multiple controllers or other angular modules.

We will take the example of our BookStore App to create an authentication service for it, and a booksFactory with factory object.

## Ionic service – authentication service

Ionic service is a special Angular construct that represents an object created using a constructor pattern, exposing certain methods abstracting app logic or integrating with the backend. Services can be injected into any module using **Dependency Injection (DI)** but its instance is created only once. Service is lazily instantiated only when any component that depends on it comes into focus. There are multiple in-built utility services such as `$http`, `$timeout`, and so on which can be used throughout the app.

A service is created using the `.service()` method available on the `angular.module` object. The first argument needs to be the name of the service and the second argument is the dependency injection array, which has the names of all dependencies, and the last element is the constructor function.

The following code explains the usage of this service:

```
angular.module('module.services', [])
.service('ServiceName',
  ['dependency1','dependency2',
   function(dependency1,dependency2) {
    this.variable1 = 1 //default value
    this.variable2 = 3 //default value
    this.method1 = function() {
      return binaryOp(this.variable1,this.variable2);
      //binaryOp can be any local method
    }
  }
]
);
```

We can model our BookStore sample app's authentication service using this construct. We will expose four given methods to this service as follows:

- `signup`: A method used to sign up a new user, taking input as a user object
- `login`: A method to log in a user and save `sessionToken` to cookies
- `logout`: A method to log a user out of a web service and remove the `sessionToken` argument

Here is the code sample for defining a service:

```
angular.module('BookStore.Services', [])
.service('UserAuth',
  ['$http','$cookies',
   function($http,$cookies) {
    this.baseApiUrl = 'http://localhost:9000/api'; // Sample
    URL to be replaced by your own API url
    this.signup = function(userObj) {
      // Making a POST $http call to /signup API call
      return $http.post(this.baseApiUrl+'/signup',userObj)
    }
    this.login = function(username,password) {
      // Making a GET $http call to /login API call
      return $http.get(this.baseApiUrl+'/login?username='+username+
        '&password='+password)
      .then(function(response) {
        if(response.data.loginSuccessful == 'true')
        {
          $cookies.put('sessionToken',response.data.sessionToken);
        }
      });
    }
  }]);
});
```

```
        }
        this.logout = function() {
            // Making a GET $http call to /logout API call
            return $http.post(this.baseApiUrl +
                'logout?sessionToken=' + $cookies.get('sessionToken'))
                .then(function(response) {
                    $cookies.remove('sessionToken');
                });
        }
    );
});
```

## Ionic factory – BooksFactory

Ionic Factory is very similar to service and can be used interchangeably with it. The only syntactical difference is that it returns an object exposing selective methods, which can be called to use the service. It is also based on a singleton pattern and the same instance is shared between different modules.

The service or factory can be used by injecting it into any controller or other service, and calling the exposed methods passing required parameters. We can use this to create a factory pattern for our app using it to manage entities or objects.

The following code explains the usage of this service:

```
angular.module('module.services', [])
.factory('ObjectFactory',
['dependency1', 'dependency2',
function(dependency1, dependency2) {
    this.objectsArray = [] //load array from server
    this.getObjects = function() {
        return this.objectsArray;
    }
}
])
;
.controller('ConsumerCtrl', ['ObjectFactory',
function(ObjectFactory) {
    $scope.objects = ObjectFactory.getObjects();
}]);
```

We can create a BooksFactory factory to implement logic regarding managing book entities in our app. We can expose the methods that can be used by multiple views or controllers. We will be exposing the following methods:

- `getBooks`: A method that can be used to get a list of books to be displayed. Certain optional arguments can be passed to the method to filter the results:
  - `Category`: Fetch books related to that category
  - `Author`: Fetch books for a specific author
  - `SortBy`: Pass the field according to which list should be sorted
- `getBookDetails`: Passing a specific book ID to get all the details regarding that book.
- `addToFavorite`: Adding a book to your favorites.

Here is the code sample for defining a service:

```
angular.module('BookStore.Services', [])
.factory('BooksFactory',
 ['$http',
 function($http) {
  var baseApiUrl = 'http://localhost:9000/api'; // Sample URL
  // to be replaced by your own API url
  var booksList = [];
  var favoriteBooks = [];
  var getBooks = function(category,author,sortBy) {
   var filters = {};
   if(category)
    filters['category'] = category;
   if(author)
    filters['author'] = author;
   if(sortBy)
    filters['sortBy'] = sortBy;
   // Making a POST $http call to /signup API call
   return $http.post(baseApiUrl+'/books',filters)
  }
  var getBookDetails = function(bookId) {
   // Making a GET $http call to /books/:id API call
   return $http.get(baseApiUrl+'/books/'+bookId);
  }
  var addToFavorite = function(bookObj) {
   if(bookObj)
    favoriteBooks.push(bookObj);
   return true;
  }
 }]
```

```
        return { getBooks: getBooks,
            getBookDetails: getBookDetails
            addToFavorite: addToFavorite
        }
    }
]
);

```

## \$resource and REST API

REST is becoming the most popular choice of design pattern for the latest web services being developed. All new platforms, frameworks, and programming languages support REST standards out of the box. REST interfaces usually involve a collection of resources with identifiers, and supporting different actions using all HTTP verbs. For example, GET requests to a /users/ resource would fetch a list of users, and POST requests to /users/ can be used to create a new user. The identifier can be used in a GET request to the URL /users/john to fetch a specific user object with a unique identifier john.

\$resource is an in-built factory that helps create a local resource object that integrates with the REST API implicitly. The resource object has in-built methods representing common methods such as save, get, delete, and so on. These methods internally interact with the API URL using the \$http object. \$resource is available only after injecting the ngResource module.

The signature for the \$resource service constructor is as follows:

```
$resource(url, [paramDefaults], [actions], options);
```

The url argument can contain dynamic parameters represented by names starting from the ':' symbol. The second argument defines the default values for the dynamic parameters expected in url. The third argument, actions, is used to map new actions to custom url. This object can contain one or multiple action fields defined by objects having properties such as method, params, url, isArray, and many more. The \$resource constructor when executed returns the object that contains the methods exposed, such as \$save, \$get, and so on, used to perform CRUD.

The following is the code example:

```
var Book = $resource('/book/:bookId', {bookId:'@id'});
var books = Book.query(function(){
    //books var get filled with books list from server
});
Book.get({bookId:123}, function(book) {

```

```
book.isNew = true;  
book.$save();  
});
```

## Demystifying mBaaS

More and more Mobile Apps are being developed at every passing second, but they also die out as fast as they appear. The average life of an app on the mobile of a user is just three months. If we spend a lot of money and effort in developing a backend for every app including the REST API, authentication service, login service, and so on, it is not justified. The mobile backend needs special services such as a multi-platform Push Notification service, analytics service, logging service, and so on. It is very tedious to develop these for each and every app you make, so certain IT companies started providing mobile backends as a service on the cloud.

The mBaaS services connect the cloud storage to Mobile Apps using custom SDKs or REST APIs. They also provide supplementary services such as Push Notification services, User Management, Social Networking services, and so on. A few popular mBaaS services are:

- **Parse:** Facebook acquired mBaaS providing JS SDK, which can be used with Hybrid Apps
- **Firebase:** Google acquired mBaaS specializing in real-time APIs for use cases such as chat, news stream, and so on
- **Azure Mobile Services:** mBaaS by Microsoft on the popular cloud platform Azure
- **Kinvey:** One of the most mature and old mBaaS providing SDKs for all platforms
- **Anypresence:** Popular mBaaS providing the code for the backend to enable enterprise-level customization

We will be discussing only two out of this list, Parse and Firebase.

## Integrating with Parse

Parse has evolved from being a simple cloud backend to a sophisticated platform having a multitude of features. Parse is maintained by Facebook itself and has excellent integration with all Facebook SDKs.

Parse has segmented its platform into three major sections:

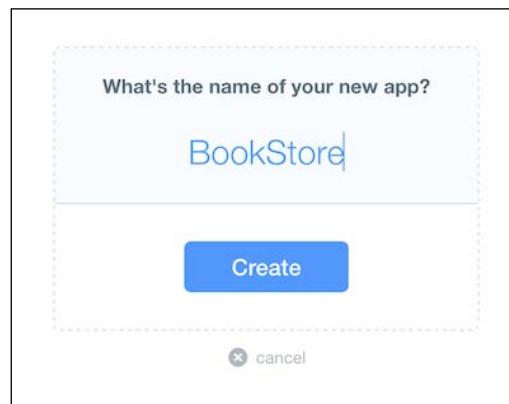
- **Core:** APIs to store and retrieve data, user management, and webhooks for writing custom logic on the cloud
- **Push:** Services enabling the management of Push Notifications for multiple platforms
- **Analytics:** Tracking any data or events for your app and users in real time

Discussion on all of these services is beyond the scope of this book, but we will explain how you can integrate Parse into your enables and leverage the platform.

The high-level steps involved in integrating your Ionic App will be discussed here.

## Step 1 – creating an app on Parse

In order to create an app on Parse, one requires to sign up for the account on [www.parse.com](http://www.parse.com). The basic account is absolutely free and provides sufficient storage and bandwidth for a prototype or the initial phase of any app. After registering on Parse, log in to your account and create a new app by giving its name as shown in the following screenshot:



## Step 2 – getting API keys

In order to get API keys that will be used to integrate SDK or REST, the API needs to be obtained from the console. After logging in to the Parse console and selecting the appropriate app on the top left, go to the **Settings** page from the top menu.

On the **Settings** page there is a menu on the left where an option for **Keys** needs to be selected. This view will contain all the keys—**REST API Key**, **App ID**, **Client Key**, and **JavaScript Key**. The important keys for us are the **REST API Key**, **App ID**, and **JavaScript Key**. These keys can be used while integrating with the Ionic App.

## Step 3 – configuring appropriate settings

In the same view, other important settings can be configured, for example, all the settings regarding user authentication such as using username and password authentication or social authentication. We can also set email settings such as reply-to email address, or define mail templates to be sent on sign up, or password reset.

The settings related to Push Notifications can be configured using the **Push** menu option on the left.

## Step 4 – integrating SDK or integrating using REST API

We will be discussing two ways in which we can integrate with Parse, either using the SDK or using the REST API directly.

### Using SDK – downloading and overview

Parse has SDKs available for each native mobile platform, iOS, Android, and Windows phone. It has SDKs for other programming languages and platforms such as PHP, OS X, Windows, .NET, Arduino, and so on. We would require only the JS SDK as we are developing a Hybrid App using JS only. The JS SDK can be downloaded from <https://parse.com/docs/downloads>.

The JS SDK is based on backbone-styled collections. The SDK does not require any external libraries and is a collection of JavaScript objects and methods. The main objects are `Parse.Object` and `Parse.User`, which are used with multiple methods. The SDK contains functionality to query data from the server, create data, upload binary files, authenticate users, and call custom server code too. The JS key needs to be set or initialized in the `run` block of our Ionic App.

The custom code written on the server is called Cloud Code and is written using JavaScript too. Most of the node modules can also be used in your Cloud Code. The Cloud Code methods can be called using the JS SDK and REST API too. The JS SDK calls can be abstracted using Ionic Services defined earlier in this chapter.

The detailed guide for the JS SDK is available at <https://parse.com/docs/js/guide>.

## Using REST API

The REST API can be integrated easily if we do not want to learn about the JS SDK. The API follows many standards and can be used with Ionic \$resource discussed in this chapter. REST API calls can also be made using a normal \$http object and abstracted using Ionic services/factories.

Along with the REST API calls, the **Application ID Key** and **REST API Key** need to be sent as the custom headers **X-Parse-Application-Id** and **X-Parse-REST-API-Key**.

We can make API calls to in-built objects such as Users, Roles, Sessions, Files, Cloud Functions, and Push Notifications using the URLs /1/<objectName>/ :<objectId> and different HTTP verbs to perform create, read, update, or delete operations.

We can use the **Core** option on the top menu and sub menu **Data** to add a class and then add rows. These classes can be managed using the URLs /1/classes/<className>/ :<objectId> with different HTTP verbs such as GET for getting a list of objects, POST to create a new single object, PUT to update specific objects, GET along with object id to get details for specific objects, and DELETE for deleting specific objects. These API URLs are perfect for using with the \$resource object as they follow REST principles. To get more details, please use the URL <https://parse.com/docs/rest/guide>.

## Integrating to Firebase

Firebase storage specializes in real-time communication. Firebase is best suitable for use cases such as chatting, news stream, collaboration apps, and so on. It also provides user management services and integration to social APIs.

Firebase stores the database in JSON format and maintains a single version only, which is synced for all the clients connected to it. The Firebase SDKs ensure syncing the clients and managing data locking. It even works offline and records your operations offline to be synced online later.

Few important facts about the firebase are as follows:

- Firebase authentication is a set of features that enable your Mobile Apps to be authenticated using SDKs only. It supports standard user- and password-based authentication as well as social auth integrations with Facebook, Twitter, Github, and Google.
- Firebase Hosting enables you to host your web apps also on Firebase's production grade cloud servers and are SSL-enabled.
- Firebase also has a JS SDK and REST API that can be used to integrate to the Ionic App in a similar way with Parse. The URL for the JS SDK is <https://www.firebaseio.com/docs/web/api/>. The detailed documentation for integrating the Firebase REST API can be found at <https://www.firebaseio.com/docs/rest/api/>.

## Summary

We have finally learned how to integrate our Ionic App with any backend and add life to our app. Ionic App will be enabled now to fetch and store data from any custom-made web service or easy-to-use mBaaS available on the market. We have seen how to create Ionic services and use them in multiple views or controllers. MbaaS seems to be a perfect choice for wiring up a quick backend to your Ionic App. Firebase and Parse are the recommended mBaaS with wide community support. In the next chapter we will learn about testing our app on real devices.



# 7

## Testing App on Real Devices

We have learnt how to develop Mobile Apps for multiple platforms and devices using Ionic Framework, but we cannot launch it unless we test it. During the development also, we should test our apps on actual devices so that we can validate the functional aspects of our app. The Mobile App will behave differently on various platforms and screen sizes and hence it is necessary to check the output on most of the popular devices.

In this chapter, we will learn how to run our apps on actual devices using different approaches. We have already seen how we test our Ionic Apps in the browser during development. We will also briefly discuss the special provision in our browsers to emulate actual devices and test our app on tens of options without the need of getting the physical devices. An individual developer cannot easily get hold of all devices such as an iPhone, Nexus, iPad, Galaxy, or Windows Tab/Phone. So, I would recommend using the option of testing your Ionic App on all the emulator types available in the browser.

But generally, as developers are very fond of devices, they might end up having or arranging one device per platform. We will be discussing different platforms and methods to test your app on real devices. The following topics will be covered in this chapter:

- Testing on browser emulators
- Ionic view app
- Making a debug build
- Remote debugging
- Testing using Ngrok

It is important to build our apps bug-free so that they are readily accepted by public app stores such as Google Play and the Apple app store. At the end of this chapter, we will briefly describe the procedure for submitting the builds of your Ionic App to these app stores.

## Testing on browser emulators

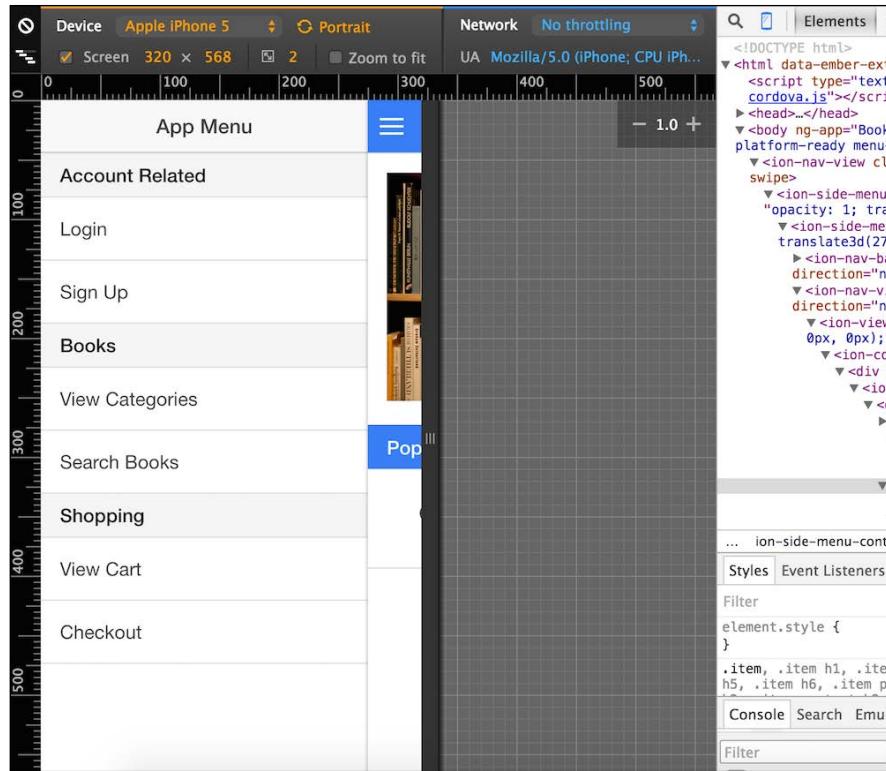
It is so blissful if we can test our Ionic App on different platforms and screen sizes without the hassle of deploying and installing it several times. Google Chrome, the most popular browser, has released a new device mode in their developer tools console from version 32 and above in the browser itself.

The device mode enables the developer to test any web URL on multiple screen sizes and platforms as Chrome emulates the browser userAgent, screen size, and resolutions. It also allows us to analyze the site performance on different network speeds by changing the throttle or network speed used for accessing that specific site. We can also simulate other device inputs such as touch, geolocation, and device orientation.

## Overview of device mode in Chrome dev tools

In order to access all the emulators we need to start the device mode from the Chrome dev tools. Opening dev tools can be done either by going to the **Chrome** menu on the top right and selecting the **Tools | Developer Tools** option or we can right-click on any page element and select **Inspect Element**.

The keyboard shortcuts for opening dev tools are *F12*, *Ctrl + Shift + I* for Windows and *Cmd + Opt + I* for Mac machines. After opening the Chrome dev tools, you have to click on the device icon to turn on the **Device Mode**. When the **Device Mode** is turned on, the icon will turn blue, as shown in the following screenshot:



On the top left, you can set the device presets. There is a drop-down menu to select the device that you want to emulate. It has a multitude of options with existing devices from popular manufacturers such as Samsung, Sony, LG, Apple, and so on. If you select a particular device, the appropriate user agent is automatically selected, touch emulation is enabled, and the screen size and resolution are set. You can also change the screen size to emulate any custom device. There is another interesting option of **Network** for network throttling. The default option selected for this drop-down is **No throttling**, which can be changed to any network speed ranging from GPRS (50 KBPS) to Wi-Fi (30 MBPS). This option can also help you test your Ionic App for offline use cases.

This tool can only be used for basic testing as it has limitations such as GPU/CPU behavior is not emulated, the device browser UI for native elements is not emulated, and system displays such as address bar are not shown. It is therefore very important to test your app on actual devices, which we will learn about in the next sections.

## Ionic view app

It can be a herculean task to set up the development environments for each platform and build apps. Ionic creators have released this excellent Ionic view app, which can be used to test your apps during the development phase without having to install it every time.

Ionic view app is integrated with the Ionic CLI. There are simple commands in the CLI that enable you to upload the app you are developing. This platform requires the developer to create an account on the `ionic.io` platform, which enables free access to this service.

The Ionic view app can be downloaded from the Google Play store and the Apple app store.

## The Ionic upload command

In order to run the `upload` command, change the directory to the Ionic Project you want to upload. From the command prompt or terminal, type the following command in the same directory position:

```
ionic upload
```

The execution of this command will lead to a prompt asking for a username and password, if not signed in already. The same username and password that were used to sign up to the `ionic.io` platform will be required. After entering the credentials, the app will be uploaded to the cloud platform and the following output will be there:

```
RahatKhannaMachine:BookStore rahat.khanna$ ionic upload
No previous login existed. Attempting to log in now.

To continue, please login to your Ionic account.
Don't have one? Create a one at: https://apps.ionic.io/signup

Email: yehtechnologies@gmail.com
Password:
Logged in! :)
Uploading app....
Saved app_id, writing to ionic-core.js...
Successfully uploaded (f2ee313c)

Share your beautiful app with someone:

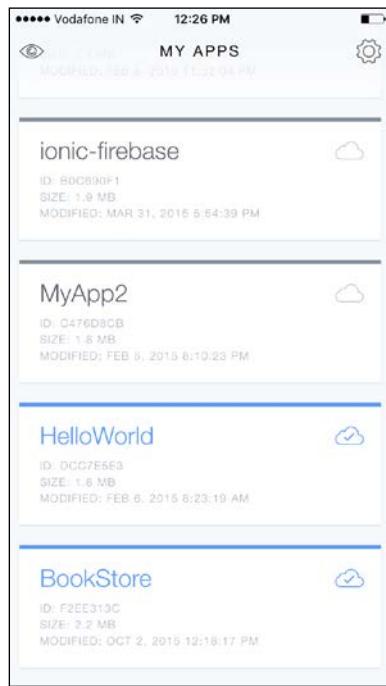
$ ionic share EMAIL

Saved api_key, writing to ionic-core.js...
```

The uploaded app can be shared with additional collaborators by using the `ionic share <email>` command. The collaborators also need to have an `ionic.io` command.

## Viewing your app

After installing the Ionic view app from the public app stores, you would need to log in using the `ionic.io` credentials. The Ionic view app has a default view to display the list of apps uploaded to your account. It would show a gray shade for the app that is in your account but not downloaded to the specific device, a blue shade for the apps that are downloaded to the specific device, and a green shade for the app that has been shared with you by someone else as shown in the following screenshot:



The Ionic view also supports Cordova device plugins such as SQLite, Camera, Device Motion, Barcode, and so on, and hence it is a perfect platform to test your apps on actual devices easily. Once you have uploaded the app, you can update the changes and test incremental versions as soon as you develop any new feature.

It also automatically detects platform-specific changes and renders your app appropriately on iOS and Android devices. You can install it on multiple devices and check it on different screen sizes.

## Making debug build

While we can use the Ionic view app to see the output of our app and share the app with others to get feedback, but we cannot debug errors during the development on the actual app using this method.

We have to approach testing on actual devices by connecting our devices to development machines using USB and deploying debug builds to enable debugging. The process for making debug builds and running them on iOS and Android are different. We will discuss the method for both in this section.

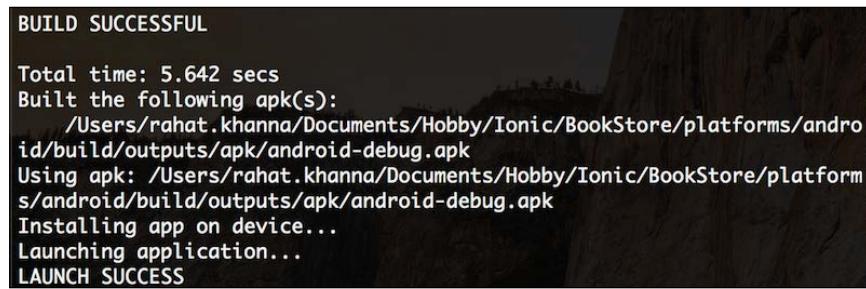
## Android debug build

In Android, the most basic step is to enable USB debugging in your Android device. The detailed steps are:

1. Enable the developer mode by going to **Settings | About Phone | Build number | Tap 7 times to become developer.**
2. After entering the developer mode you have to go to **Settings | Developer Options | USB Debugging.**
3. Now, you should connect your device to the development machine.
4. A dialog will appear on your device to allow the specific RSA key – **Please press the Ok button.** If you do not see this dialog, to ensure your system is connected, go to the `Android SDK tools` folder and type the command `adb devices`.
5. Now the device has been connected to the machine and enabled for debugging.

We will leverage the Ionic platform to make a debug build. Please go to the directory for your Ionic project. Please add Android and iOS using the `ionic platform add android ios` command. You can also use only a single platform name to add the respective platform separately. After the specific platforms are added, we can use the `ionic run [android|ios]` command to run our Ionic App on the specific device that is attached to our device. This will initiate the downloading of the required build dependencies for the first time, but from the next time, your build will be updated in your device instantly.

We will see the message **Launch Success** when the app is launched on your device, as follows:



```
BUILD SUCCESSFUL

Total time: 5.642 secs
Built the following apk(s):
  /Users/rahat.khanna/Documents/Hobby/Ionic/BookStore/platforms/android/build/outputs/apk/android-debug.apk
Using apk: /Users/rahat.khanna/Documents/Hobby/Ionic/BookStore/platforms/android/build/outputs/apk/android-debug.apk
Installing app on device...
Launching application...
LAUNCH SUCCESS
```

In order to see debug logs, please open Eclipse and Logcat or go to the `Android SDK platform-tools` folder and type in the commands `adb shell` and `logcat`.

## iOS debug build

In iOS, making a debug build run on an actual device is a bit tricky. Apple has restrictions for installing and running apps on devices, which is managed by the Apple Developer Program. In order to run your app on an actual device you need to obtain a \$99 annual fee license.

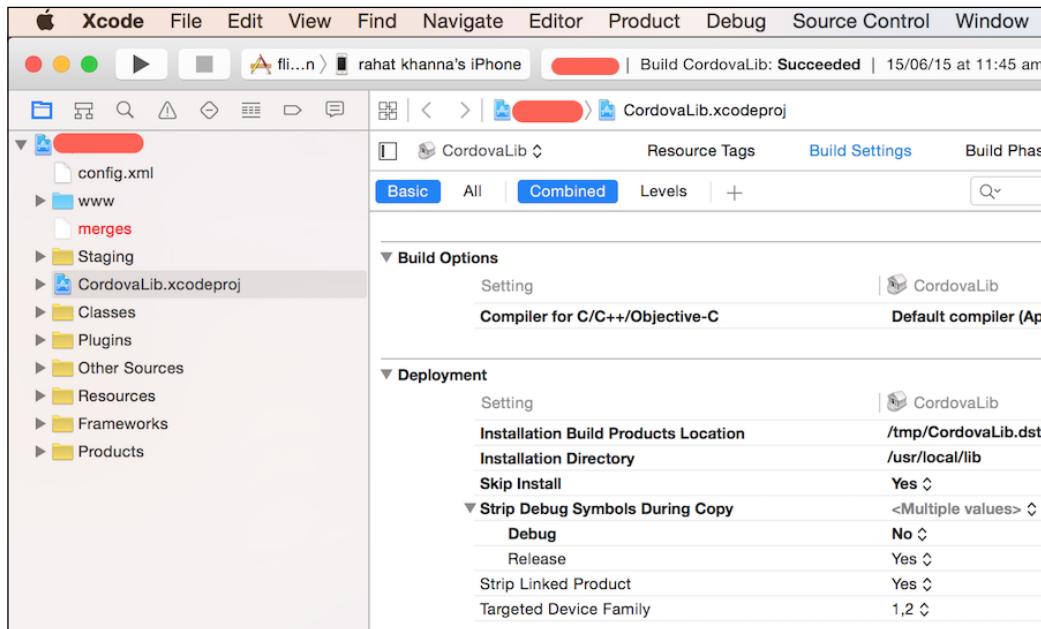
After buying an Apple Developer license, we need to obtain a provisioning profile. You have to sign in to the Apple Developer portal and create a developer provisioning profile. While you create a provisioning profile, you have to give a unique app ID (bundle identifier) for the app.

You have to install the provisioning profile after downloading it by double clicking on it. Go to the folder `platforms`, then to the `ios` folder and open the Xcode Project. Now, you should update the bundle identifier to match the id you used while creating the provisioning profile.

## *Testing App on Real Devices*

---

You can now use Xcode to select your device from the top panel and then click on the play (run) button. The Xcode should look like the following screenshot:



## **Remote debugging**

There are times when the Mobile Apps being tested do not need to be connected to your development machine. Hybrid Apps mostly run JavaScript code, so the debugging is done mostly on the JS console. The remote debugging can be done using two ways, one using Chrome's remote web inspector tool and the other using a free tool called jsconsole.

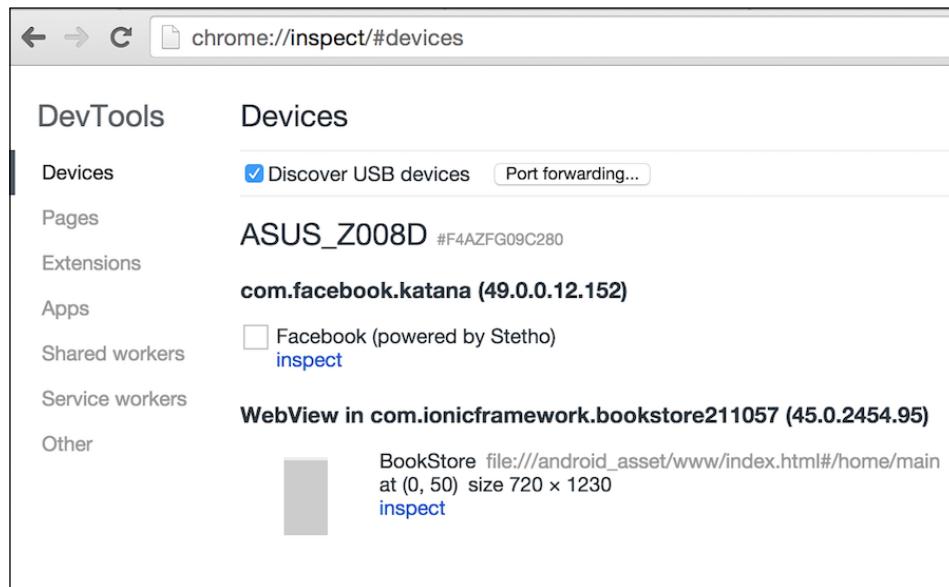
## **Remote debugging using Chrome dev tools**

Chrome enables a special remote debugging protocol for iOS and Android devices. It helps you see the logs remotely for your Ionic App running in a browser or as an app on your devices. In Android, the remote debugging is enabled for Chrome and also WebViews in apps. In iOS, remote debugging is enabled for your Ionic App running in Safari.

## Android debugging

The basic requirements are Chrome 32 or later on the development machine, a USB cable to connect to the machine, and the latest Chrome for Android on the device itself. The steps to be followed are given here:

- USB debugging should be enabled on your device
- In order to discover a device on desktop Chrome, open the URL `chrome://inspect` on the desktop browser and check the **Discover USB devices** option
- Every connected device will be shown with its open tabs
- In order to inspect a specific tab or device, click on the **inspect** link on the respective option as shown in the following screenshot:



## iOS debugging

On iOS devices, Safari has the capability of remote debugging with a feature of Safari web inspector. The steps required are as follows:

1. Open Safari on the iOS device.
2. It is required to connect your iOS device to the system.

3. Safari needs to be installed on the development machine too. Open the Safari browser.
4. Go to the **Develop** menu option on the top, and look for your device's name.
5. There will be a list of tabs open on your iOS device Safari browser that you can remote debug on your development machine.

## Remote debugging using jsconsole.com

jsconsole is an online website that emulates the JavaScript command line tool. It enables you to bridge any remote WebView or browser and remotely receive logs and send JavaScript commands to the remote device. It is very useful for cases where you want to debug issues for remote users who cannot be with you.

The basic requirement is you have to inject a script tag generated online on [jsconsole.com](http://jsconsole.com) and put in the app or mobile web app that you want to debug. The steps that must be followed are:

1. Go to [jsconsole.com](http://jsconsole.com).
2. Create a session using the command `:listen` on the website.
3. You will get a script tag with a unique URL that you have to copy and paste into `index.html` of your Ionic App.
4. Any calls to `console.log` from your Ionic App running on any remote device will now be logged onto the website [jsconsole.com](http://jsconsole.com).

The following screenshot shows the output of the preceding steps:



A screenshot of a web browser window showing the jsconsole.com interface. The address bar at the top displays "jsconsole.com/?%3Alisten". The main content area is a terminal-like window with the following text:  
|  
> :listen  
**i** Creating connection...  
**i** Connected to "F92E9BF8-A584-4E47-BBFE-97E7469BF81E"  
<script src="http://jsconsole.com/remote.js?F92E9BF8-A584-4E47-BBFE-97E7469BF81E"></script>  
> :help

## Testing using Ngrok

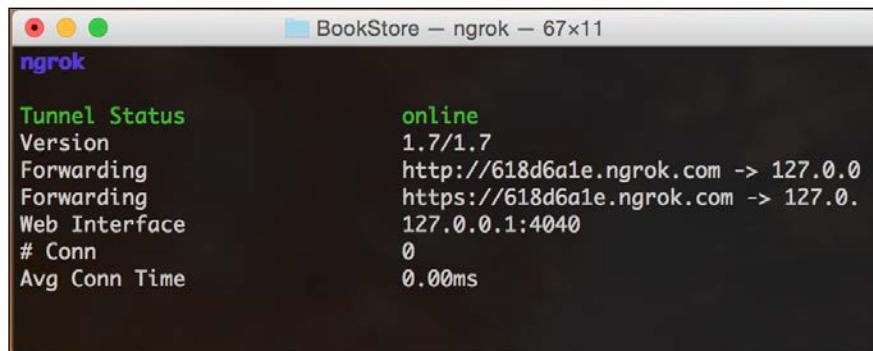
We have learned a lot about remote debugging but there is a problem such as hosting your Ionic App to a public location where any device on the Internet can access it. Also, sometimes you have to integrate your Ionic App to some local APIs and then test it on an actual device.

It would be very tedious and cumbersome to upload your backend and Ionic App to a public-facing server every time you want to test some functionality. Ngrok is an excellent tool that enables you to expose your local host server on any port to the Internet world using a public URL.

Ngrok acts as a secure tunnel between your local machine and a public URL, which people can access on any device. You can use this public URL in the methods given previously in this chapter to remotely debug your Ionic App.

Ngrok is very easy to use, you have to download it from the Internet. It will be downloaded as an archive, unzip it, and save it. You can also install using the `npm` command, `npm install -g ngrok`. Now open your Ionic App locally in the browser by using the command `ionic serve`.

Your app will open up in the browser on `http://localhost:<port_number>`. If we assume that the port number is 8100, then you can start the Ngrok service by using the command `ngrok 8100` in the command line prompt [windows] or terminal [mac]. Ngrok will output a random public URL such as `s2323h4.ngrok.com`. You can use this URL to access your local Ionic App from any remote device and keep on testing or debugging. The following screenshot depicts the screen output when `ngrok` command is executed:



```
ngrok
Tunnel Status      online
Version           1.7/1.7
Forwarding        http://618d6a1e.ngrok.com -> 127.0.0.1:4040
                  https://618d6a1e.ngrok.com -> 127.0.0.1:4040
Web Interface    127.0.0.1:4040
# Conn            0
Avg Conn Time   0.00ms
```

## **Summary**

In this chapter we have learned different ways to test our Ionic App during development. We have also learnt different ways to share our app with co-collaborators or colleagues to get feedback. A Mobile App requires a lot of feedback from various users before publishing it to the public. In the next chapter we will learn how to integrate Cordova plugins into your Ionic App and leverage hardware features of your device such as camera, geolocation, and so on.

# 8

## Working with Cordova Plugins – ngCordova

In the previous chapters we have learnt how to develop beautiful apps using Ionic components and integrate them into different backends. In this chapter we will learn how to leverage hardware features in Ionic Apps using Cordova plugins. A Mobile App has evolved today from more than just displaying or taking input for information. Mobile Apps have become more powerful in all walks of life, such as mediums for taking pictures from the camera, giving commands via voice, scanning barcodes of products, finding locations and services around us, biometric authorizations, and even helping us to keep fit. In a Hybrid App, utilizing the device capabilities is a bit tricky, but Cordova makes it easy.

ngCordova is an amazing open source library that has Angular wrappers around open source Cordova plugins. This library has a collection of services and extensions that were initially developed by the Ionic team but are driven by the community these days. In this chapter we will be covering the following topics:

- Introduction to Cordova plugins
- Integrating ngCordova to the Ionic App
- Important plugins
  - Camera plugin
  - Push notifications
  - Geolocation
  - Contacts
  - Network
  - Device sensors
- Custom Cordova plugin development

We will be learning about how to use the basic features of these plugins so that we can develop special features in our Ionic App using these.

We do not need to write any native code for any platform. The plugin includes the code for each native platform and also a JS file to expose the methods for consumption in JavaScript.

## Introduction to Cordova plugins

Cordova's main purpose is to package web code into a Native App, but the second most important job is to create a bridge between the web(JS) code and the native code. Cordova ensures that Hybrid Apps harness the power and performance of native code perfectly. Cordova plugins utilize this power and enable us to use the specific device features in JS easily.

Cordova from version 3 and above decided to expose all device APIs as plugins. The Cordova plugin is a reusable set of code that contains the native code as well as JavaScript code. We know that our web app runs inside a WebView control in Ionic/Cordova Apps. The JavaScript code of the plugin exposes methods that will call a bridge/interface to invoke a native method for the respective platform such as iOS(Objective C) and Android(Java).

The contents of a Cordova plugin package are:

- Native code for each platform
- A common JavaScript library
- A manifest file called `plugin.xml`

The folder structure for the Cordova plugin project is:

```
// Main Plugin Folder
- plugin.xml
- src/
- android/
// Java (Android SDK) code
- ios/
// Objective-C code
- www/
// Javascript library
```

The Cordova library has the basic device API plugins developed, and can be integrated with your Ionic App easily. Cordova also maintains a list of third-party open source plugins available, which can be downloaded and added to your app too. If there is some functionality for which no plugins can be found, we will discuss the process for creating a custom plugin at high level.

## Plugin management

In order to manage such a large list of in-house and third-party plugins, Cordova uses a plugin management system called Plugman. Plugman is a command line tool that enables features such as searching for a plugin, adding a new plugin, removing an existing plugin, or changing plugin configurations.

## Integrating ngCordova to Ionic App

ngCordova is a JS library that acts as an Angular wrapper to all Cordova plugins. Angular has its own architectural style and hence Cordova plugins do not fit directly into an Ionic App. ngCordova makes it super easy for Ionic developers to call JS methods for Cordova plugins and process the data input.

ngCordova is available as a JS file, which can be added to your project using Bower if it is being used as a dependency management system. The bower command to install the specific ngCordova dependency into your Ionic App is as follows:

```
$ bower install ngCordova
```

The ngCordova JS file can also be directly downloaded from the GitHub website, <https://github.com/driftyco/ng-cordova/>. There will be two files in the dist folder, ng-cordova.js and ng-cordova.min.js. You can include the reference to one of these files in the index.html file of your Ionic App using the following code:

```
<script src="lib/ngCordova/dist/ng-cordova.min.js">
<script src="cordova.min.js">
```

The ngCordova JS file has a module named ngCordova, which contains services exposing functionalities for most of the popular Cordova plugins available online. We need to inject the ngCordova angular module into our Ionic App root module. We have to add the following code to app.js:

```
angular.module('myApp', ['ngCordova']);
```

In order to use Cordova plugins in JavaScript, we need to wait for the deviceready event, which notifies us about the completion of loading of native code. The ngCordova plugin calls will not work before the device ready event is fired.

The Ionic platform provides an `Ionicready` event hook that is always executed after a device ready event. Any ngCordova code should be written inside this `$ionicPlatform.ready` event hook as follows:

```
// Named Callback Function
function executePluginCode() {
    $cordovaPlugin.someFunction().then(success, error);
}
$ionicPlatform.ready(executePluginCode);
```

Although ngCordova is used to invoke native code and handle responses, the actual heavy-weight work is being done by the Cordova plugins themselves. We have to manage the Cordova plugins from the Cordova command line tool or the Plugman tool separately. The list of Cordova plugins supported by ngCordova is given on its website. The code to add a new plugin is:

```
cordova plugin add <plugin_name>
```



All the Cordova plugins do not work on browsers or emulators so please test your Ionic App on real devices using the methods mentioned in previous chapters for those plugins.



## Important plugins

We will discuss a few important plugins that are required in most of the apps. These plugins are available for both iOS and Android, but we will discuss the common JS code here as the native code is abstracted from the developer.

## Camera plugin

This plugin enables taking pictures and videos from the camera using your Ionic App and saving it to the local storage. This is available in the ngCordova module as the `$cordovaCamera` service. We have to use dependency injection to inject this service to any controller or service we want to use this in.

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add cordova-plugin-camera
```

The `$cordovaCamera` service exposes one method `getPicture(options)` to invoke the native camera API. The object options passed to this method define the settings and the behavior of the action to be completed from the app. The options parameter is passed as an object containing the following optional properties/fields:

Options property name	Type	Description
quality	Number	Quality of the image, range of 0-100
cameraDirection	Number	Selection of camera – Back:0, Front-facing:1
sourceType	Number/Enum	Setting the source for the picture, values supported: Camera . PictureSourceType . [PHOTOLIBRARY   CAMERA   SAVEDPHOTOALBUM]
mediaType	String	Choosing the type of media to select from
encodingType	Number	Type of image to save JPEG:0, PNG:1
correctOrientation	Boolean	Correct captured images in case of wrong orientation
destinationType	Number	Set the destination for the image, values supported: Camera.DestinationType . [FILE_URI   DATA_URL]
allowEdit	Boolean	Allow simple editing of image before selection
saveToPhotoAlbum	Boolean	Decide whether to save image to photo album on device
targetWidth	Number	Width to scale the image (pixels)
targetHeight	Number	Height to scale the image (pixels)

This method call returns the object with image data. It would contain the image data URL or the image file URL based on the options passed. In our BookStore sample app we can use this to click the photo of the book cover. The code to show the image using the data URL scheme is:

```
angular.module.controller('MyBookPictureCtrl', function($scope,
$cordovaCamera,$ionicPlatform) {
  $scope.bookImage = { src: '/img/dummyBookImage.jpg' };
  $scope.takeBookPhoto = function () {
    $ionicPlatform.ready(function () {
      var pluginOptions = {
        sourceType: Camera.PictureSourceType.CAMERA,
        encodingType: Camera.EncodingType.JPEG,
```

```
        saveToPhotoAlbum: false,
        destinationType: Camera.DestinationType.DATA_URL
    };
$cordovaCamera.getPicture(pluginOptions).then(function(picData) {
    $scope.bookImage = "data:image/jpeg;base64," + picData;
}, function(err) {
    // error
});
});
}
});
```

This is the code for saving the image to the local storage on the device. You will get the file location and data URL, which can be stored locally or on any remote server using web services:

```
angular.module.controller('MyBookPictureCtrl', function($scope,
$cordovaCamera,$ionicPlatform) {
    $scope.bookImage = { src: '/img/dummyBookImage.jpg' };
    $scope.takeBookPhoto = function () {
        $ionicPlatform.ready(function () {
            var pluginOptions = {
                sourceType: Camera.PictureSourceType.CAMERA,
                encodingType: Camera.EncodingType.PNG,
                destinationType: Camera.DestinationType.FILE_URI
            };
            $cordovaCamera.getPicture(pluginOptions).then(function(picURI) {
                $scope.bookImage = picURI;
            }, function(err) {
                // error
            });
        });
    }
});
```

In our BookStore App, we can add a feature to click a picture of a new book and store it for displaying in the book listing. The camera plugin can be used in a wide variety of use cases such as storing any scene, object state, saving memories, and so on.

## Push Notifications

Mobile Apps need not poll the server for regular updates as they can leverage the push design pattern using Push Notifications technology. Implementing Push Notifications in Mobile Apps is tedious as it requires a middleware server integrating to cloud servers of all platforms such as the **Apple Push Notification Services (APNS)**, **Google Cloud Messaging (GCM)**, and the **Windows Push Notification Service (WNS)**.

We will not discuss the implementation of this middleware server. This server can be replaced by cloud push service providers such as Parse, Kinvey, and Ionic. Ionic push is very easy to configure and will work perfectly with the ngCordova push plugin. We will be discussing the ngCordova push plugin API to register, unregister, and receive Push Notifications. The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add https://github.com/phonegap-build/PushPlugin.git
```

After adding the plugin and ngCordova library, we can use the push plugin as the `$cordovaPush` service available in the ngCordova module. We have to inject this `$cordovaPush` service into any controller or service we want to use it.

In order to enable Push Notifications for a specific Mobile App on a device, the app needs to register its unique device ID. The method to register a device is:

- **Method Signature:** `register(config)`
  - **Returns:** Object (it will contain user info and device token (iOS) or `registrationId` (Android))
  - **Parameter:** The parameter `config` object to be passed can have the following fields:

Property name	Type	Platform	Description
badge	Boolean	iOS	Whether the Push Notification should have a badge.
sound	Boolean	iOS	Whether the Push Notification will have a sound alert.
alert	Boolean	iOS	Whether the Push Notification will show an alert.
senderID	String	Android	String representing a unique ID extracted from the Google cloud console specific to a project.

The second method to unregister a device has the following details:

- **Method Signature:** `unregister(options)`
  - **Returns:** Promise (it returns a promise that can be attached to a success handler or error handler)
  - **Parameters:** Options parameter to be passed is optional

The code example is as follows:

```
// Registering for Push Notifications
angular.module('BookStoreApp', ['ngCordova'])
.run(['$cordovaPush',function($cordovaPush) {
    var gcmConfig = {'sender_id':
        'sender_id_from_google_console'};
    $cordovaPush.register(gcmConfig).then(function(result) {
        // Result object will have a RegistrationId which you
        need to send to your middleware server
    }, function(err) {
        // Invoked if Error Received
    });
    $cordovaPush.unregister(gcmConfig).then(function(result)
    {
        // It will successfully unregister your device/app with GCM
    }, function(err) {
        // Invoked if Error Received
    });
}]);
```

The Push Notifications are received using angular events in the ngCordova library. The event name for the Push Notification is `$cordovaPush:notificationReceived`. The event can be listened using the `$on` method on `$rootScope`. The event will receive two arguments, event and notification object. The sample code for listening to Push Notifications and receiving them is as follows:

```
angular.module('BookStoreApp', ['ngCordova'])
.run(['$cordovaPush', '$rootScope',function($cordovaPush,$rootScope)
{
    $rootScope.$on('$cordovaPush:notificationReceived',
    function(event, pushNotif) {
        if (pushNotif.alert) {
            navigator.notification.alert(pushNotif.alert);
        }
        if (pushNotif.sound) {
            var alert = new Media(event.sound);
            alert.play();
        }
    });
}]);
```

```
        }
        // notification.title & notification.message will be the main
        fields
    });
});
});
```

Push Notifications can also have custom payload, depending on the server sending it. We can process the Push Notification and implement custom logic on receiving the push.

## Geolocation

The Geolocation Cordova plugin is used to get the current location of your device. It can also get continuous location of your device tracking the movement of the device. The service `$cordovaGeolocation` available in the module `ngCordova` can be used to integrate the Geolocation plugin to your Ionic App.

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add cordova-plugin-geolocation
```

The methods available in this service are given as follows:

- **Method Name:** `getCurrentPosition(options)`
  - **Returns:** Promise (the success handler of promise receives a position object containing coordinates including latitude, longitude, altitude, speed, and so on)
  - **Parameter:** The options parameter can have three fields: `timeout` (number) – the number of milliseconds to wait for a response, `maximumAge` (number) – the number of milliseconds defining the age of the cached response that will be accepted, and `enableHighAccuracy` (Boolean) – guiding the plugin to provide the best results

The code example is as follows:

```
$scope.findBooksNearby = function() {
  var configOpts = {timeout: 2500, enableHighAccuracy:
    true};
  $cordovaGeolocation
    .getCurrentPosition(configOpts)
    .then(function (pos) {
      var lat = pos.coords.latitude;
      var long = pos.coords.longitude;
      // Pass lat & long to your service for finding books
      nearby
```

```
    }, function(errorObj) {
        // Invoked if Error Received
    });
}
```

- **Method Name:** `watchPosition(options)`

- **Returns:** `watchId` Promise (every time the position changes, the success handler of promise receives a position object containing coordinates including latitude, longitude, altitude, speed, and so on.)  
The `watchId` object returned will be used in the next method to stop watching position changes.
- **Parameter:** The options parameter can have three fields: `timeout` (number) – the number of milliseconds to wait for a response, `maximumAge` (number) – the number of milliseconds defining the age of the cached response that will be accepted, and `enableHighAccuracy` (boolean) – guiding the plugin to provide the best results.

- **Method Name:** `clearWatch(watchID)`

- **Returns:** Promise (standard promise registering success or error callback).
- **Parameter:** `watchID` returned from the `watchPosition` method.

The code example is as follows:

```
$scope.movementCoords = [];
var watchMovement;
$scope.trackMovement = function(){watchMovement =
$cordovaGeolocation.watchPosition(configOpts);
    watch.then(null,
        function(err) { // error },
        function(positionCoords) { // position object
            $scope.movementCoords.push(positionCoords);
        });
}
$scope.clearTracking = function () {
    $cordovaGeolocation.clearWatch(watchMovement)
    .then(function(result) { // Success in stop recording
        movement },
        function (error) { // Error invoked if unable to stop});
}
```

## Contacts

A contacts plugin is used to manage the contacts on the device. It is available for both iOS and Android. It has two major methods – one for saving a new contact and the other for fetching contacts based on some parameters. The service `$cordovaContacts` is available under the `ngCordova` module for managing the contacts plugin.

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add cordova-plugin-contacts
```

The methods available under this service are:

- **Method Name:** `save (contact)`
  - **Returns:** Promise (standard promise registering success or error callback)
  - **Parameter:** The `Contact` object should be passed as a parameter, the contact object can have these common properties: `displayName` (string), `name` (string), `phoneNumbers` (array), `emails` (array), and `birthday` (date)
- **Method Name:** `find (filterOptions)`
  - **Returns:** Promise (standard promise registering success or error callback) – the success callback returns the list of contacts fetched
  - **Parameter:** The `filterOptions` object, which can contain the properties – `filter` (string[`searchTerm`]), `multiple` (boolean), `fields` (array[to be searched]), and `desiredFields` (array[returned fields])

The code example is as follows:

```
$scope.addNewFriend = function(userInfo) {
    // Saving New Contact from userInfo object
    $cordovaContacts.save(userInfo).then(function(result) {
        // New Contact is written to device.
    }, function(err) {
        // Error invoked if unsuccessful.
    });
}

$scope.findFriends = function() {
    var opts = { filter : 'searchTerm',
                multiple: true,
                fields: [ 'displayName', 'name' ]
                desiredFields: ['id','name'] }
```

```
    };  
    $cordovaContacts.find(opts).then(function(filteredContacts)  
    {  
        $scope.friendsSearchResult = filteredContacts;  
    });  
}
```

## Network

The network device plugin helps in identifying the network on which the device is connected and listening to network change events. The `$cordovaNetwork` is a service of the ngCordova module, which exposes multiple methods and events for managing the network.

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add cordova-plugin-network-information
```

The methods available under this service are:

- **Method Name:** `getNetwork()`
  - **Returns:** The `Connection` object (this property determines the connection state and connection type)

The possible connection types can be:

Type	Description
<code>Connection.UNKNOWN</code>	Unknown connection
<code>Connection.ETHERNET</code>	Ethernet connection
<code>Connection.WIFI</code>	Wi-Fi connection
<code>Connection.CELL_2G</code>	Cell 2G connection
<code>Connection.CELL_2G</code>	Cell 3G connection
<code>Connection.CELL_2G</code>	Cell 4G connection
<code>Connection.CELL</code>	Cell generic connection
<code>Connection.NONE</code>	No network connection

- **Method Name:** `isOnline()`
  - **Returns:** Boolean (true if network is online)
- **Method Name:** `isOffline()`
  - **Returns:** Boolean (true if network is offline)

There are two events that this service exposes, which help us in listening to changes in the network such as notifying when the device goes online or offline

- **Event Name:** `$cordovaNetwork (online)` – this is fired when the device goes online
  - **Returns:** Event (Angular Event Object), Network State (Network state/type of the connection)
- **Event Name:** `$cordovaNetwork (offline)` – this is fired when the device goes offline
  - **Returns:** Event (Angular Event Object), Network State (Network state/type of the connection)

The code example is as follows:

```
angular.module('BookStoreApp', ['ngCordova'])
.run(['$cordovaNetwork', '$rootScope', function($cordovaNetwo
rk, $rootScope) {
    $rootScope.networkType =
        $cordovaNetwork.getNetwork();
    $rootScope.isDeviceOnline =
        $cordovaNetwork.isOnline();
    $rootScope.isDeviceOffline =
        $cordovaNetwork.isOffline();

    // Handling online event & updating $rootScope flag
    $rootScope.$on('$cordovaNetwork:online', function(event,
networkState) {
        $rootScope.networkType = networkState.type;
        $rootScope.isDeviceOnline = true;
    });

    // Handling offline event & updating $rootScope flag
    $rootScope.$on('$cordovaNetwork:offline', function(event,
networkState) {
        $rootScope.networkType = networkState.type;
        $rootScope.isDeviceOffline = true;
    });
}]);
```

## Device sensors

The mobile device has a lot of sensors such as accelerometer, gyrosensor, and compass, which are being used by a variety of Mobile Apps in the health, transportation, gaming, and so on, sectors. We can leverage these device sensor inputs in our Ionic App using device services in the ngCordova module.

## Device motion

The device motion is managed using the \$cordovaDeviceMotion service of the ngCordova module. This plugin provides access to the device accelerometer. It reports the change in speed or movement of the device in three dimensions (x, y, and z).

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add cordova-plugin-device-motion
```

The methods available under this service are:

- **Method Name:** getAcceleration()
  - **Description:** It fetches the current acceleration along the x, y, and z axes
  - **Returns:** Object (contains the x, y, z coordinates and timestamp)
- **Method Name:** watchAcceleration(options)
  - **Description:** It fetches the device's current acceleration at regular intervals
  - **Returns:** Promise WatchID (success handler object contains the x, y, z coordinates and timestamp)
  - **Parameters:** Options (the options object should have a frequency property whose value should be in milliseconds defining the interval after which a new acceleration will be fetched)
- **Method Name:** clearWatch (watchID)
  - **Description:** It clears the watchAcceleration regular calls
  - **Parameters:** watchID (the id returned by watchAcceleration() call)

The code example is as follows:

```
$scope.accelerationData = [];
var watchId =
$cordovaDeviceMotion.watchAcceleration(configOpts);
watchId.then(null,
  function(error) {
    // An error occurred
  },
  function(response) {
    $scope.accelerationData.push({
      x:response.x,
      y:response.y,z:response.z, timestamp:response.timestamp
    })
  });
$cordovaDeviceMotion.clearWatch(watchId)
.then(function(result) {
  // Stopped listening to motion changes
}, function (error) {
  // Error in stopping the event listener
});
```

## Device orientation

The device orientation is managed using the `$cordovaDeviceOrientation` service of the `ngCordova` module. This plugin provides access to the device compass sensor. It helps in identifying the direction or heading that the device is pointed to. It measures the heading in degrees from 0 to 355.9, where 0 is north.

The command line to be executed using the Ionic/Cordova CLI is:

```
cordova plugin add org.apache.cordova.device-orientation
```

The methods available under this service are:

- **Method Name:** `getCurrentHeading()`
  - **Description:** It fetches the current compass heading reading
  - **Returns:** Object (contains `magneticHeading`, `trueHeading`, `headingAccuracy`, and `timestamp`)

- **Method Name:** watchHeading(options)
  - **Description:** It fetches the device's current heading at regular intervals
  - **Returns:** Promise WatchID (success handler object contains magneticHeading, trueHeading, headingAccuracy, and timestamp)
  - **Parameters:** Options (the options object should have a frequency property whose value should be in milliseconds defining the interval after which a new acceleration will be fetched)
- **Method Name:** clearWatch(watchID)
  - **Description:** It clears the watchHeading regular calls
  - **Parameters:** watchID (the id returned by watchHeading() call)

The code example is as follows:

```
$scope.headingData = [];  
var watchId =  
$cordovaDeviceOrientation.watchHeading(configOpts);  
watchId.then(null, function(error)  
{  
    // An error occurred  
},  
function(response) {  
    $scope.headingData.push({  
        magneticHeading:response.magneticHeading,  
        trueHeading:response.trueHeading,  
        accuracy:response.headingAccuracy,  
        timeStamp:response.timestamp  
    });  
});  
$cordovaDeviceOrientation.clearWatch(watchId)  
.then(function(result) {  
    // Stopped listening to compass changes  
, function (error) {  
    // Error in stopping the event listener  
});
```

## Custom Cordova plugin development

There are multiple third-party open source plugins available that can be integrated to your Ionic App along with the ngCordova library. If we do not find the required functionality plugin, we can develop a custom Cordova plugin and use it along with the ngCordova library.

We have to write the native code for each platform we want to port our plugin on. The native coding part will be standard according to the procedures followed in the respective native platform. Cordova helps in exposing the same functionality using the native hybrid bridge. In your JavaScript file, you should call the following method to invoke the native functionality:

```
cordova.exec(successCallback, failureCallback, service, action,  
[args]);
```

The previous code will execute/invoke the action method on the service class on the native side. If the native code completes successfully, it will call the `successCallback` or, if it fails, it will call `failureCallback`. The detailed process to develop a custom plugin is beyond the scope of this book.

## Summary

In this chapter we have learned how custom Cordova plugins help in integrating the native device features in a Hybrid App. We have also learnt how to use the ngCordova library in our Ionic App to integrate the device plugins and use them in our controllers and views. We have also seen the utility of the most important device plugins and API services available under the ngCordova library.



# 9

## Future of Ionic

Ionic has evolved from a framework to an ecosystem. Ionic started initially as a JS framework for developing a Hybrid Mobile App front end but it has come a long way and there is more to come. Ionic has become a platform to empower next gen Mobile Apps through web technologies and cloud services. In this chapter we will learn about these cloud services, which you can use to empower or augment the Ionic Apps you have built using this book. Also, we will discuss the future of the Ionic JS framework into its version 2, which will be based on Angular 2 involving a re-haul of the architecture using new technologies such as ES6/2015.

Ionic has launched the initial alpha version of v2 and it will take some time for it to stabilize. We will discuss the new features and migration strategy for v2. Ionic also has launched a portal, [ionic.io](http://ionic.io), for managing all the cloud services it provides. We will discuss an overview of all these services. In this chapter we will discuss the following topics:

- Ionic cloud services
  - Ionic Creator
  - Ionic Market
  - Ionic Push
  - Ionic Deploy
  - Ionic Analytics
  - Ionic Package
  - Ionic Lab
- Ionic v2
  - New features
  - Migrating to v2

These cloud services are available to developers for free in alpha version, so we should try all of them in order to learn about their true benefit. Also, Ionic v2 is the next succession path for Ionic v1 apps developed, so we should know what's new and also devise a migration path for the same.

## Ionic cloud services

Mobile Apps have evolved from simple games or calculator apps to become more like communities and platforms for social engagement. All popular Mobile Apps must have features such as social sharing, collaboration, and Push Notifications these days. These features, if developed for each Mobile App, can account for a lot of effort. Ionic has launched this cloud platform [ionic.io](https://ionic.io), which provides cloud services to be integrated into existing Ionic Apps using SDKs for each platform. A signup is required for using all these products and services on [ionic.io](https://ionic.io).

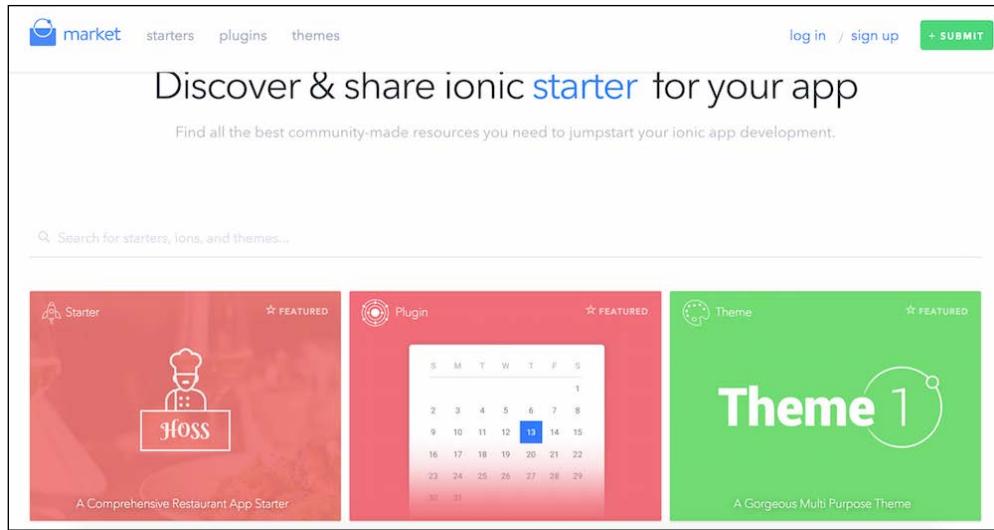
## Ionic Creator

Ionic Creator is an online GUI wizard to create Ionic Apps using a drag and drop utility. We have already discussed Ionic Creator in detail in *Chapter 3, Start Building Your First Ionic App*. Ionic Creator can also be used to design a few of the views for your app.

Ionic Creator can manage multiple views and also manage the links between different views. It provides an easy-to-use interface to drag and drop components or UI elements from the side panel. Almost all types of components or elements are available such as button, image, button bar, popups, and even custom HTML. The code for the views designed using Ionic creator can be exported and downloaded as a ZIP file. It will hold HTML, CSS, and JavaScript files, which can be integrated with your existing Ionic App too.

## Ionic Market

Ionic has set to change the industry with a robust Hybrid App development framework. Ionic has been instrumental in creating a strong community of developers creating amazing Ionic Apps and components. Ionic Market is a platform to empower this growing community of Ionic developers. Here is a screenshot for the Ionic Market web page:



Ionic Market is available for developers to showcase their code and distribute it either for free or for a fee. Ionic Library involves the basic components and features that can be used to develop Mobile Apps. Generally, developers require much more complex reusable components and features to develop their apps rapidly. It is very hard to find awesome work done by other developers by searching online through their distributed codebases.

Ionic Market enables developers to upload and submit listings for their starter templates, Ionic plugins (Ionic re-usable code), themes (Scss/CSS files). Anyone with an `ionic.io` platform can buy or sell starters, plugins, or themes on this open exchange platform. Developers can also link their listings to any third-party marketplace where they would be selling their components primarily.

Ionic Market does not even charge anything or take any cut, except the payment platform charges or any taxes involved in the buying/selling of these digital pieces. I would recommend you to list your Ionic Apps/components if you want to earn while being an awesome Ionic developer.

## Ionic Push

Push Notifications are always tricky to implement for any Mobile App. Apart from using the push plugin in a Hybrid App, we also require a backend that can send Push Notifications to cloud servers of multiple platforms such as iOS, Android, and Windows. Ionic Push saves a lot of development effort and helps developers get to the market faster without any investment in a backend.

Ionic Push integrates very well with the push plugin and the ngCordova library. Simple configurations are required to make it work and Ionic push provides an easy-to-use console to manage the sending of Push Notifications. We will describe the basic steps to enable `ionic.io` and Ionic push to be used with your Ionic App:

- **Step 1:** In order to use the `ionic.io` SDK in our app, we need to add a web-client component using the Ionic CLI tool only. We should also add the push plugin at the same time using the following code:

```
ionic plugin add phonegap-plugin-push  
ionic add ionic-platform-web-client
```

- **Step 2:** Ionic CLI also provides commands for managing the Ionic io modules. We have to initialize `ionic.io` for our app. We can run the following command from the root folder of our Ionic App:

```
ionic io init
```

- **Step 3:** You can now use the Ionic push angular services to write code to register a device for Push Notifications and also listen to events for receiving Push Notifications. The `ionic.service.core` and `ionic.service.push` modules are required to be injected for this.

- **Step 4:** Now, we need to configure our app to receive Push Notifications. This needs a special setup and configurations for iOS and Android. iOS Push Notifications are mediated by **APNS (Apple Push Notification Services)** and Android by **GCM (Google Cloud Messaging)**. The detailed guide for setting up is available at the links <http://docs.ionic.io/docs/push-ios-setup> (iOS) and <http://docs.ionic.io/docs/push-android-setup> (Android).

We can also test Push Notifications using Ionic push without even doing these configurations for all the platforms using the dev mode. These are fake notifications generated by the `ionic.io` module.

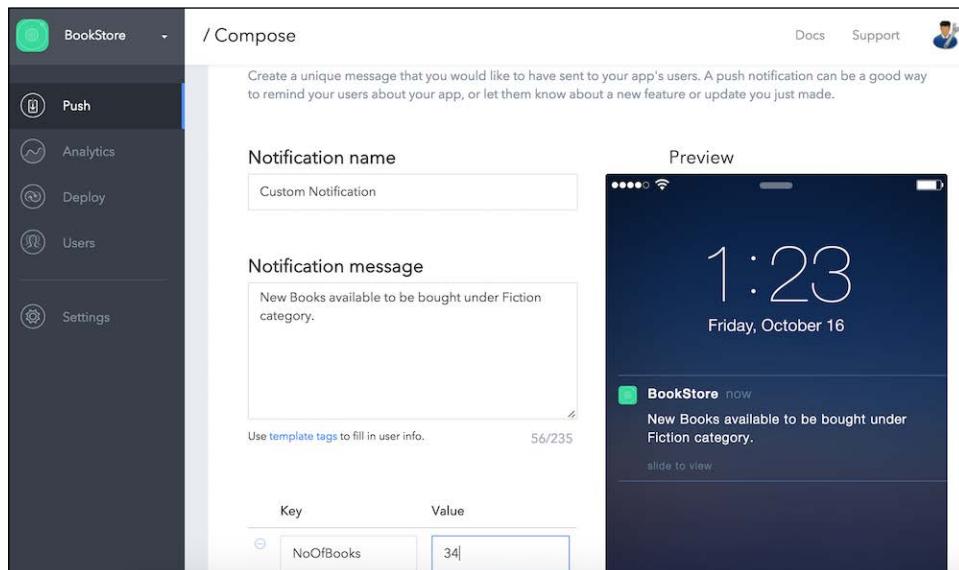
- **Step 5:** Integrating your backend for listening to push registrations and sending Push Notifications. There are two ways in which you can perform these actions. You can either use the `ionic.io` cloud console to see the devices registered and even send Push Notifications using a simple interface, or you can use their REST API to call these methods from your own backend server and listen to push registrations and send Push Notifications.

You can register your backend webhook URL to receive the push registration events using the following command:

```
ionic push webhook_url http://your-server-url
```

It will be hit with a HTTP POST request with the device token and extra keys to identify the user every time a new device is registered using Ionic Push.

The REST API endpoint to send a push is <https://push.ionic.io/api/v1/push> [HTTP POST]. The details about headers and payload to be passed to this request can be found at the URL <https://push.ionic.io/api/v1/push>. Ionic push enables you to send an instant Push Notification or even schedule one for later delivery to all users or specific user IDs. Here is a screenshot for the dashboard for composing a Push Notification:



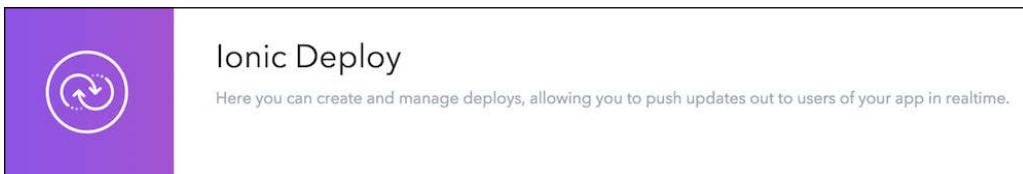
## Ionic Deploy

Once your apps are on the public app stores such as the Apple app store and Google Play store, the update cycles are pretty lengthy. It is a tedious job to submit a new update to the app store and go through the complete review process. It takes at least a few days or even weeks for your updated app to be released online. Your users also have to go to the respective app store and download the complete updated version of the app.

Ionic Deploy (in alpha state) solves this problem by giving the capability to the developers for updating their apps on the fly. It enables on-demand improvements and enhancements done on the app to reach the users immediately. It does not require recompilation and updating the build anywhere. It also supports rollback to a previous version instantly. Ionic deploy can also be used for collaborating with designers and testers during the development of the app.

## Using Ionic Deploy

In order to leverage the benefits of Ionic Deploy, you can upload your Ionic App to cloud servers using the `ionic upload` command. This would upload your web assets of the Ionic App to a cloud server. The uploaded content will always be augmented with a unique version number and a timestamp. Ionic deploy also lets you create separate builds for separate people using something called Deploy Channels. Any update can be sent to a specific Deploy Channel and will be sent to the apps installed with that specific deploy channel. Default channels named `production`, `stage`, and `dev` already exist and can be used immediately. Alternatively, the developer can decide to create any custom deploy channel also. For more details on Ionic deploy please go to the URL <http://docs.ionic.io/docs/deploy-overview>. The following screenshot will appear:

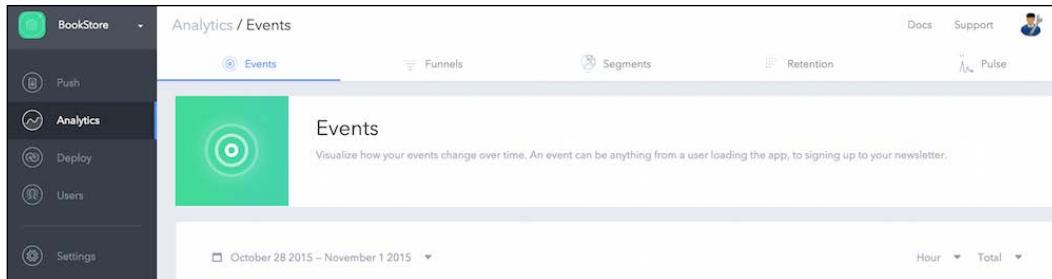


## Ionic Analytics

Ionic Analytics is the latest alpha state cloud service launched under the `ionic.io` platform, which enables users to track important metrics and usage data for their app users. It is very important for developers to know about important insights into the usage of their app so that they can plan for enhancements or updates. Data and facts are always essential for making the right decisions and this service from Ionic helps you take wise decisions for your Ionic Apps. Existing analytics platforms do not have a lot of tools for Hybrid Mobile Apps.

Ionic Analytics takes care of everything for Hybrid Mobile App needs and requires no other analytics suite to coexist. Ionic Analytics fits right in because it knows the internal architecture and working of an Ionic App including routing, controllers, and components. Ionic Analytics uses events to record data from user interactions. The event consists of an event name and the associated data for it.

The `$ionicAnalytics` service of the `ionic.service.analytics` module is used to send event data automatically as soon as the user interacts with your app. There are a lot of built-in events that are automatically sent but are configurable using the `$ionicAutoTrackProvider`. The `$ionicAnalytics` service also enables developers to register custom events and send custom data for these events in your Ionic App code using the `track(name, dataObject)` method. Here is a screenshot for the dashboard to view the events:



## Ionic Package

We have already learned in previous chapters how difficult it is to set up development environments for different iOS and Android platforms and then package the app. In order to save developer's efforts from these hassles, the Ionic Package cloud service has been released under `ionic.io`. It is a cloud service that automatically packages the app on the go and provides ready to release packages for iOS and Android platforms. This cloud service does not even require a Mac machine for iOS builds, but the Apple Developer license needs to be procured as mentioned in one of the previous chapters.

Ionic package enables developers to build dev or production packages using simple commands under the `ionic` CLI command set. These packages can be submitted to a public app store or directly installed on a friend/collaborator's device. The sample command to make a build for a specific platform is:

```
ionic package build android --profile PROFILE_TAG
```

Details regarding more Ionic Package options can be found at <http://docs.ionic.io/docs/package-overview>.

## Ionic Lab

We have seen that there are a lot of commands and configurations for developers to remember. In this era of automation and self service, Ionic has come up with a beautiful GUI tool called Ionic Lab for Mac and Windows to help developers create Ionic Apps easily. It includes all the features to manage your Ionic Apps on a development machine end to end. In my opinion it is a life savior for people who do not like terminal but love user experience.

Ionic Lab provides a user-friendly way to perform all the operations we have seen Ionic CLI commands for in this book. It also provides a live preview for your app for testing it. The overview of capabilities for Ionic Lab are:

- **App creation:** You can create a new app with a click of a button using any of the starters available
- **Running app:** You can run any app on a simulator or device from the Ionic Lab GUI
- **Make builds:** You can click a button to generate a build for any platform
- **Preview app:** Ionic Lab has a live preview feature for testing your app
- **Upload app:** An easy button to upload the app to the `ionic.io` server for it to be viewed on devices using Ionic view is also provided
- **App logs:** An interface to view the logs for your Ionic App is also provided

## Ionic v2

The Ionic team has recently announced the alpha version of Ionic 2 at an Angular Connect conference in London in October 2015. The Ionic team has worked closely with the Angular 2 team to align the Ionic 2 perfectly with the Angular 2 release. This has also given the opportunity to improve the base architecture of Ionic and move towards more performant Hybrid Mobile Apps. Ionic 2 has an improved routing and navigation methodology and also an optimized way of integrating with native APIs.

## New features

We will be discussing the new features in Ionic v2 briefly in this section.

### Angular 2, ES6, and Typescript

As Angular 2 has decided to leverage ES6/ES2015, the latest version of JavaScript that supports classes and other exciting features, Ionic 2 apps will also be written with ES6 or Typescript. Typescript is an extended version of ES6 with the support for types. We can leverage classes, ES6 modules, decorators, arrow functions, and block scopes.

## **Abstracted annotations**

Although Ionic 2 is built on top of Angular 2, the Ionic team has baked in a nice abstractions that would help you overcome the fear of Angular 2. For example, the new annotation @App built by the Ionic team over the existing Angular 2 annotations. It will be super easy for any developer who knows just the basics of Angular 2 to make apps using Ionic 2.

## **Material design**

Ionic has developed special theming for the Android platform to support the popular material design conceptualized by Google.

## **Enhanced Native Integration**

Ionic 2 boasts about direct integration of more native APIs into the platform so that no external plugins are required to build Mobile Apps. It would make it easy for Ionic Apps to harness the power of the device they are running on.

## **Powerful Theming**

Ionic v1 lacked an important feature such as theming. Ionic v2 solves that problem and improves the theming apart from the basic platform-specific themes. Ionic v2 makes it easy to customize themes according to your own brand guidelines or color preferences. The theming will be managed by SASS stylesheets only.

## **Improved navigation and routing**

Ionic has re-engineered the routing and navigation pieces in v2 to make it more app-like. The navigation enables you to link to even any sub view-like modal of a view, component, and so on. It will bring a more app-like experience to the users and enable deep linking in your Ionic v2 Apps.

## Migration to v2

Ionic v1 will be supported by the Ionic team for a long time and we will not need to convert all our production apps to v2. The Ionic team will release enhancements and new features for Ionic v1 to make the migration path to v2 much easier.

A few important points regarding migration:

- Learning the basics of Angular 2 are very important for migrating your apps
- Learning all the new cool constructs in ES6 will also help in the ease of migration
- All the components and plugins from v1 are still there in v2 with the underlying concepts; most of them have not changed
- The views and controllers from v1 have been merged into one using `@Page` annotation and ES6 classes
- There are new entities called components that can be navigated to directly
- In your Ionic v1 apps, try to use the controller as syntax to make the migration very easy at a later stage
- Convert your code into ES6 or Typescript as valid JavaScript is valid Typescript, so that your app will not break even if you change only a small part

## Summary

We have learned about the new cloud offerings from Ionic as a complete platform and how we can leverage them to make Ionic Apps rapidly and release them onto the market. We have also learned about the uber cool features of Ionic v2 and how we can plan to migrate our apps in the future.

Web developers now can develop fully featured Hybrid Mobile Apps using Ionic. In this book, we have learned how to use different components and services to add amazing user experience to our Ionic Apps. We can also leverage the native device features to enhance our Ionic Apps. We should also empower our Mobile Apps with mBaaS for providing a strong backend. The next step for Ionic developers should be to design and develop new components and share them with the online Ionic community.

# Module 2

## Ionic Framework By Example

Build amazing cross-platform mobile apps with Ionic,  
the HTML5 framework that makes modern mobile  
application development simple



# 1

## First Look at Ionic

Before we begin this book, it is very important that we understand just exactly what we are dealing with. The best way to understand this is by having a short history on mobile development, in general, and understand how tools like Ionic help mobile developers create beautiful mobile apps.

### The beginning

The year 2006 saw the beginning of the smartphone era with the launch of the iPhone by Apple. By 2008, Google had launched its answer to Apple's iOS operating system. This new operating system was called **Android**, and by 2010, it was clear that smartphones running iOS and Android dominantly covered the mobile ecosystem. Fast forward to today, the dominance of iOS and Android is not so different even though Windows for mobile by Microsoft has made some gains on the mobile front. It is fair to say that Android, iOS, and Windows make up the majority of the ecosystem with the first two at the forefront by a large margin.

The launch of the smartphone era also gave birth to the concept of mobile applications. Mobile apps are the medium by which we deliver and obtain most of our content on mobile phones. They are great and everyone with a smartphone pretty much has a number of apps downloaded on their devices to perform specific actions or achieve specific goals. This was massive for developers, and the software vendors also provided tools that enabled developers to create their own third-party mobile apps for users. We refer to these applications, built using the tools provided by the software vendors, as **native mobile applications**.

## The problem

As great as mobile apps are, there is a small problem with how they are developed. Firstly, for each mobile development platform, the software vendor provides its own unique set of tools to build applications for its platforms. We know these tools as SDKs. The following table shows how each platform differs in terms of tools and SDK options to create native mobile apps for their ecosystems:

Operating system	SDK	Programming language
iOS	iOS SDK	Objective-C/Swift
Android	Android SDK	JAVA
Windows for mobile	Windows SDK	.NET

To make a clear statement, we are not trying to downplay the use of native tools. As noted earlier, native tools are great but come with a great cost and time constraint. Firstly, you are unable to build the same app for different platforms with the same set of tools. For the Android version of your app, you will need a team of skilled android developers. For the iOS version of your app, you will need a team of Objective-C or Swift developers to create the iOS version of the same app. Also, there is no code sharing between these two teams, meaning that a feature developed on one platform will have to be completely developed on the other platform again. This is highly inefficient in terms of development and very time consuming.

Another problem is that because you are hiring two separate teams that are completely independent of one another even though they are both trying to create the same thing, you are left with a growing cost. For example, if you decided you wanted to create a Windows for mobile version of your mobile app, you will need to recruit another team of .NET developers and they will have to build everything present on the other existing platforms from scratch since they cannot reuse any of the already built tools.

For a company like Facebook, which makes revenue in the billions, it might make sense to go down the native path as cost and talent for native development would probably not be a part of their concern. However, for the most part, not everyone building or trying to build a mobile app is a company like Facebook. Most people want to get a simple, great, powerful app out there as quick as possible. Furthermore, some of these people want to use their preexisting skill set to build apps for multiple platforms without having to learn new programming languages.

Before mobile applications, web apps ruled the world for the most part. We had more people developing for the web technologies consisting mostly of HTML, CSS, and JavaScript. One great thing we got used to with the web was that it was platform independent. This meant that as long as you had a browser application on any device, you were able to interact with any web application without any problem.



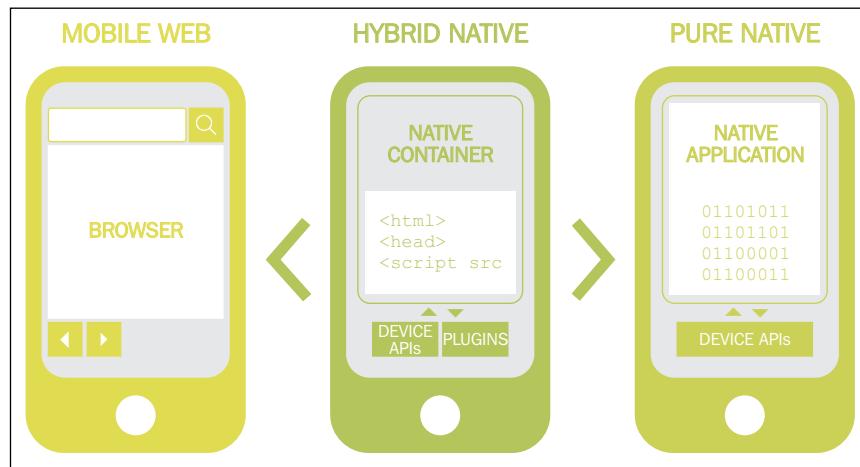
So when mobile apps came, it was a big change for most web developers because with mobile apps, each platform was self-dependent, and apps made for one platform would not work for another platform.

## Apache Cordova

Apache Cordova is a technology that lets any web application be packaged as a native mobile application while also providing access to device features. Thanks to Adobe and the open source community, this technology has seen great growth over the years and more and more apps are being built with Cordova. The apps built with Cordova are commonly referred to as **hybrid applications**. But what is a hybrid app?

A hybrid application in the context of Cordova is actually a web app that lives within the thin container of a native mobile application.

In reality, the only difference between a native mobile app and a web application in terms of what they can do is the fact that the native mobile app has access to the device hardware features.



In truth, a hybrid app is actually a native app that serves up a web application on the phone's web view. It behaves and acts like a normal application and has complete device access, thanks to Cordova.

However, the main advantage that Cordova has over native development techniques is that you only have to maintain one code base, and can use that same code base to build applications for multiple platforms. This was and still is the main selling point of Cordova to date, as with this technique you are provided with a big cost and time saving advantage.

## Early hybrid apps

When the first few hybrid apps started rolling out, there were a couple of problems that seemed to persist. The first problem was that a lot of people complained that these hybrid applications did not have the same user interface and user experience as native mobile apps. The major reason for this was that when building a native app, all the building blocks are already provided for you. For example, we have things like pre-provided animations, swipe gestures, tabs, and so on. Hybrid apps failed to provide similar features like these because on the web, all we have is HTML, CSS, and JS. There is no pre-provided component for things like animations, gestures, and tabs.

This meant that pretty much anyone trying to build a hybrid app had to build these features from scratch. This was not very good and different people had different implementations of the same features. As a result, a lot of applications that were built by the hybrid way were not so visually appealing. What we needed was a framework that was centrally maintained that provided us with all the tools we needed to build features that native apps had with web technologies.

## What is Ionic?

Ionic is a framework that lets you build hybrid mobile applications with web technologies like HTML5, CSS, and JavaScript. But that is not where it stops with Ionic. Ionic provides you with components that you can use to build native-like features for your mobile applications. Think of Ionic as the SDK for making your Hybrid mobile application. Most of the features you have on a native app such as modals, gestures, popups, and many more, are all provided to you by Ionic and can be easily extended for new features or customized to suit your needs.

Ionic itself does not grant you the ability to communicate with device features like GPS and camera; instead, it works side-by-side with Cordova to achieve this. Another great feature of Ionic is how loosely coupled all its components are. You can decide to use only some of Ionic on an already existing hybrid application if you wish to do so.

The Ionic framework is built with AngularJS, which is arguably the most well-tested and widely-used JavaScript framework out there. This feature is particularly powerful as it gives you all the goodness of Angular as part of any Ionic app you develop. In the past, architecting hybrid applications proved to be difficult, but with Angular, we can create our mobile applications using the **Single Page Application (SPA)** technique. Angular also makes it really easy to organize your application for the development and working across teams while providing you the possibility of easily adding custom features or libraries.

## Short history of Ionic

Before we dive in, first let's revisit what we already know about hybrid applications and how they work. Remember that a hybrid mobile application is simply a web application that runs in a web view, within a thin native wrapper environment.

Also remember that native apps came with already built components that enabled you to create beautiful user interfaces for mobile applications. Since hybrid apps used web technologies, there was no SDK or components provided for creating mobile UIs. The Ionic team saw this problem and created a solution in the form of the Ionic framework. The Ionic framework provides UI components to build beautiful hybrid applications.

## **Features of Ionic**

Ionic provides you with a lot of cool neat features and tricks that help you create beautiful and well functioning hybrid apps in no time. The features of Ionic come under three categories:

- CSS features
- JavaScript features
- Ionic CLI

## **CSS features**

To start off, Ionic comes stock with a great CSS library that provides you with some boilerplate styles. These Ionic CSS styles are generated with **SASS**, a CSS preprocessor for more advanced CSS style manipulation.

Some of the cool CSS features that come built-in with Ionic include:

- Buttons
- Cards
- Header and footers
- Lists
- Forms elements
- Grid system

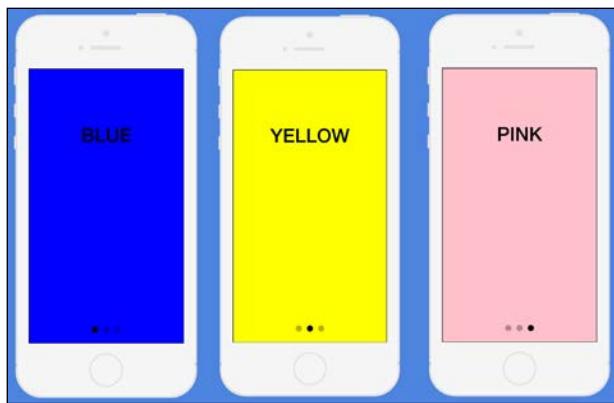
All these features and more, are already provided for you and are easily customizable. They also have the same look and feel that native equivalents have so you will not have to do any editing to make them look like native components.

## JavaScript features

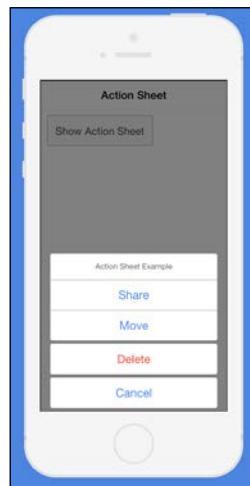
The JavaScript features are at the very heart of the Ionic framework and essential for building Ionic apps. They also consist of other features that let you do things from under the hood like customize your application or even provide you with helper functions you can use to make developing your app more pleasant. A lot of these JavaScript features actually exist as HTML custom elements that make it easy to declaratively use these features.

Some of these features include:

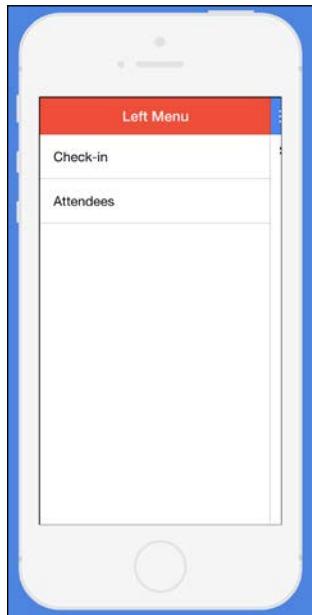
- Modal
- Slide box



- Action sheet



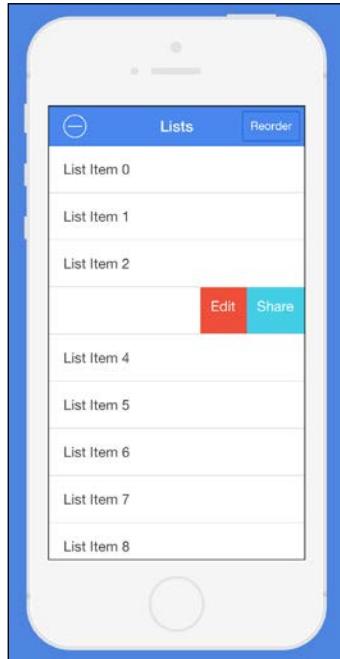
- Side menu



- Tabs



- Complex lists



- Collection repeat

All the JavaScript features of Ionic are built with Angular, and most can be easily plugged in as Angular directives. Each of them also perform different actions that help you achieve specific functions and are all documented in the Ionic website.

## The Ionic CLI

This is the final part that makes up the three major arms of the Ionic framework. The Ionic CLI is a very important tool that lets you use the Ionic commands via the command line/terminal. It is also with the Ionic CLI that we get access to some Ionic features that make our app development process more streamlined. It is arguably the most important part of Ionic and it is also the feature you will use to do most actions.

Ionic CLI features let you do the following:

- Create Ionic projects
- Issue Cordova commands
- Development and testing
- Ionic splash/Icon generator

- Ionic labs
- SASS
- Upload app to Ionic view
- Access `Ionic.io` tools

The Ionic CLI is a very powerful tool and most of the time, it is the tool we will be using throughout this book to perform specific actions. This is why the first thing we are going to do is set up the Ionic CLI.

## Setting up Ionic

The following steps will give a brief of how to setup Ionic:

1. **Install NodeJS:** To set up Ionic, the first thing you will need to do is to install NodeJS on your computer so you can have access to **Node Package Manager (NPM)**. If you already have node installed on your computer, you can skip this step and go to step 2. To install NodeJS on your computer, perform the following steps:
  1. Go to [www.nodejs.org](http://www.nodejs.org) and click on the latest stable version for your computer. That should download the latest version of NodeJS on your computer. Don't worry if you are on Mac, PC, or Linux, the correct one for your operating system will be automatically downloaded.
  2. After the download is finished, install the downloaded software on your computer. You might need to restart your computer if you are running Windows.
  3. Open up the terminal if you are on Mac/Linux or the Windows command line if you are on a Windows machine. Type the command `node -v` and press *Enter*.

You should see the version number of your current installation of NodeJS. If you do not see a version number, this might mean that you have not correctly installed NodeJS and should try running step 1 again.

2. **Install Ionic CLI:** The next step is to use NPM to install the Ionic CLI.
  1. Open a new terminal (OS X and Linux) or command-line (Windows) window and run the following command: `npm install ionic -g`. If you are on Linux/OS X, you might need to run `sudo npm install ionic -g`. This command will aim to install Ionic globally.

2. After this has finished running, run the command `ionic -v` on your terminal/command line and press *Enter*.

You should see a version number of your Ionic CLI. This means that you have Ionic installed correctly and are good to go. If you are on a Windows machine, you might need to restart your machine to see the version number appear.

If you did not see a version number, then you do not have Ionic installed correctly on your machine and should do step 2 again.

## Summary

In this chapter, we started off by getting to know a bit of background about mobile applications in general. We learned how native mobile applications work, how they are built with native SDKs, and how each platform is built with a completely different set of tools without any resource sharing between them all. We then went ahead and discussed briefly about Apache Cordova and how it aimed to solve the problem of cross-platform development.

We then discussed exactly what Ionic means and what problems it aims to solve. We also got to discuss the CSS, JS, and Ionic CLI features of the Ionic framework lightly.

In the next chapter, we will be creating our very first Ionic application with the Ionic CLI, and we will create a nice to-do list style application with some great Ionic features.



# 2

## To-Do List App

In this chapter, we will be diving headfirst into Ionic and will be using a lot of the Ionic CLI tool. We will create our first Ionic application and add some basic Ionic features to our app. We will also get to run our app for the first time using Ionic and will debug our app in Chrome. We will finish this chapter by creating a to-do list application with Ionic. This application will simply let us add items to our app and also provide us a way of deleting these items or marking them as done.

### Creating our first application

Creating a new project with Ionic is actually a very pain-free experience with the Ionic CLI. There are different ways to create a new Ionic project but the easiest and more standard technique is to use the Ionic templates. This is by far the easiest way, and it let us use any of the three standard templates provided by Ionic.

These templates include:

- **The blank template:** This creates a new project with some boilerplate code to help you get set up with a blank application
- **The tabs template:** This is the same as the first but instead of a blank application, you get an application with a tabbed design
- **The side menu template:** This creates a new application with a side menu design and some boilerplate

We will be using each of these in this book at some point of time. For now, we are going to start with the first and create a brand new Ionic project using the blank template. Before we move on, let's have a look at the command that the Ionic CLI uses to create a new application:

```
ionic create [Name Of App] [template]
```

The `create` command for the Ionic CLI allows us to provide two parameters, the first being the name we want our app to be called. This first parameter will also be the name given to the folder that gets generated with our files. The second parameter is the template name. As discussed earlier, there are three template styles. You can either pass in blank, tabs, or side menu as a parameter to represent the type of template you want your app to be generated with.

## Creating our to-do list app

We are going to create our to-do list application. We are going to use the blank template to do this. We will be calling our app `todo` for the sake of consistency. To create the `todo` app, go ahead and run the following command:

```
ionic start todo blank
```

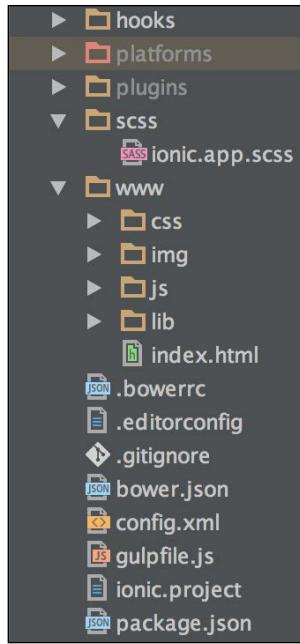
This command will create a new blank Ionic application called `todo`. When this command has finished running, enter the project of your application via the command line by running the following command:

```
cd todo
```

To further explore our newly created `todo` app, open the `todo` app folder in your favorite IDE.

## The Ionic workflow

When you create a new Ionic project, there are a couple of folders and files that come as stock as part of the generated project. Your directory should look similar to what is seen in the following screenshot:



The structure you see is pretty much the same as in every Cordova project, with the exception of a few files and folders. For example, there is a `scss` folder. This contains a file that lets us customize the look and feel of our application and will be covered in detail in later chapters. There are also the `platforms` and `plugins` folder. The `platforms` folder, in most cases is auto-generated, but we will not be covering them in this book so you can ignore them for the time being.

You will also notice that in your `www/lib` folder, there is a folder called `ionic` that contains all the required files to run Ionic. There are `css`, `fonts`, `js`, and `scss` folder.

- `css`: This folder contains all the default CSS that come with an Ionic app.
- `fonts`: Ionic comes with its own font and Icon library called **Ionicons**. This Ionicons library contains hundreds of icons, which are all available for use in your app.

- `js`: This contains all the code for the core Ionic library. Since Ionic is built with Angular, there is a version of Angular here with a bunch of other files that make up the Ionic framework.
- `sass`: This is the folder that contains SASS files that are used to build the beautiful Ionic framework CSS styles. Everything here can be overwritten easily in order to make your app feel a bit more customized and we will discuss how you can do this in *Chapter 7, Customizing the App*.

If you have a look at the root folder, you will see a lot of other files that are generated for you as part of the Ionic workflow. These files are not overly important now, but let's have a look at the more important ones in the following list:

- `bower.json`: This is the file that contains some of the dependencies acquired from the bower package manager. The browser dependencies are resolved in the `lib` folder as specified in the `bowerrc` file. This is a great place to specify other third-party dependencies that your project might need.
- `config.xml`: This is the standard `config` file that comes along with any Phonegap/Cordova project. This is where you request permissions for device features and also specify universal and platform-specific configurations for your app.
- `gulpfile`: Ionic uses the Gulp build tool, and this file contains some code that is provided by Ionic that enables you do some amazing things. We will use some features of this file in *Chapter 7, Customizing the App*, when we do some customization tasks.
- `ionic.project`: This is a file specific for Ionic services. It is the file used by the Ionic CLI and the `ionic.io` services as a place to specify some of your Ionic-specific configuration. We will use some of the features of this file when we use the Ionic view app in *Chapter 3, Running Ionic Apps*.
- `package.json`: This is a file used by node to specify some node dependencies. When you create a project with the Ionic CLI, Ionic uses both the Node and Bower Package Manager to resolve some of your dependencies. If you require a node module when you are developing Ionic apps, you can specify these dependencies here.

These files are some of the more important files that are by default a part of a project created with the Ionic CLI. At the moment you do not need to worry too much about them, but it's always good to know that they exist and have an idea about what they actually represent.

## In-depth look at our project

Before we go ahead and do any development, it is imperative that we understand how to actually add features to our app and where to do this. There are two files in particular that we are going to pay great attention to:

- `index.html`: This file is the entry point of your application in terms of what you actually see. It is a normal HTML page with some boilerplate code based on the blank Ionic template. If you pay close attention, you will see some custom HTML tags such as `<ion-pane>`, `<ion-header>`, and `<ion-content>`. These custom tags are actually Ionic components that have been built with Angular, and for now, you need not worry about what they do as we will be discussing this shortly. A closer look at the `<body>` tag will also reveal the attribute `ng-app=starter`. This is a custom attribute provided by Angular, which we use to provide the name of the main module of an angular application.
- `app.js`: This file lives in the `js` folder, and this is the file that contains the main module of our application. In Angular, modules provide us a way to create isolated chunks of code that our application uses. The main module is the module that actually gets loaded to our application when it starts. Think of the main module as the entry point of our application. If you take a closer look at the `app.js` file, you will see how we create the module and specify its name as `starter`:

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

.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();
    }
  });
})
```



### Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You will also see that it takes a second parameter, an array which contains one string called `ionic`. In Angular, this array is used to provide the names of any module that our application depends on. So, just like we specified the name of our main module, `starter` to the `ng-app` attribute in the `index.html` file, we specify a list of modules that our main module relies on, in this case, `ionic`.

You would have also noticed a `run` function in the `app.js`. This function is the function that fires as soon as our app is ready and all our dependent Angular modules and factories have loaded. The `run` function is a great place to do little pieces of tidying up that you want done as soon as your application begins.

These two files are the ones you need to worry about as they are the two main files we will be working with in order to build our to-do list application.

## Envisioning our app

It is always good to do a small bit of wireframing before you build any application. This enables you to understand how the app will work and how it will probably look just before you actually begin to code the app. The following screenshot is a rough wireframe of what our `todo` app will look like:



Our `todo` application allows a user to simply enter any task they want added to their to-do list. Think of this app as a mini diary where you put in things you want done later. In this section, we will only be building the very basic feature of our `todo` app, and we will only be allowing the user to add new items. In later chapters, as we learn more about Ionic, we will be adding more complex features like using a complex list and also letting the user edit, remove, and even archive to-do list items.

## Building our todo app

To get started with building our `todo` app, we will need to further break down what we want to achieve into smaller steps. The first thing we need to do is to create the UI for our application.

### Creating the UI

The first thing we are going to do to get started with building our `todo` application is building the user interface. We are going to build the input form and the button that will add the `todo` item currently typed in the input. After we have written the code for this, we will add the markup for the list where we want any entered `todo` item to be displayed. I have already compiled this markup for you in the following code:

```
<div class="list">
  <div class="item item-input-inset">
    <label class="item-input-wrapper">
      <input type="text" placeholder="enter todo item">
    </label>
    <button class="button button-small">
      Add
    </button>
  </div>
</div>
<ul class="list">
  <li class="item">

  </li>
</ul>
```

From the preceding code, you can see the skin of our user interface ready. We have an input that receives what we want entered into our to-do list. We have an HTML unordered list that will be placed where our to-do list items will be situated. You can see some classes on some of our elements. These are actually classes from the auto-generated Ionic CSS styles that come as part of any Ionic project.

## The code

Since we have written the user interface for our application, we will also need to write the Angular code to enable it to work. What we need to do is to create an array that will hold the list of `todo` items and also create a function that will add a `todo` item into this list anytime we click the **Add** button we created earlier. We will achieve this all by creating an Angular controller in our main module and insert all this logic into it. I have already written this code and you can copy it and get it into your project from the following:

```
.controller('TodoController', function ($scope) {
  $scope.todos = [];
  $scope.todoModel = {};
  $scope.todoModel.todo = '';
  $scope.addTodo = function () {
    $scope.todos.push($scope.todoModel.todo);
    $scope.todoModel = {
      todo: ''
    };
  };
})
```

From the preceding code, you can see that we have created a controller called the `TodoController`. Within this `TodoController`, we have a `todos` array. This is the array that will hold all our `todo` items. We also have a `todoModel` object that is an empty object that will hold our entered `todo` item. Lastly, we have a function called `addTodo` that adds the current value in our `todoModel` object to our `todos` array and then sets the value of our current `todoModel` object to an empty string so we can type from scratch again.

## Wiring things up

Now that we have created our user interface boilerplate code and also written our code for it, it is time to wire the two together and dictate what gets to appear where:

```
<ion-content ng-controller="TodoController">
  <div class="list">
    <div class="item item-input-inset">
      <label class="item-input-wrapper">
        <input type="text" placeholder="enter todo item"
               ng-model="todoModel.todo">
      </label>
      <button class="button button-small" ng-
             click="addTodo() ">
        Add
    </div>
  </div>
```

```
        </button>
    </div>
</div>
<ul class="list">
    <li class="item" ng-repeat="todo in todos track by
        $index">
        {{todo}}
    </li>
</ul>
</ion-content>
```

If you have a look at the preceding code, you will see that the UI code now looks a bit different. Firstly, we have associated our `<ion-content>` element with our `TodoController`. This is done in order to create a binding context, meaning any variable within the `TodoController` is now available for data binding to all its descendants. Secondly, you will also notice that our input now has a new `ng-model` attribute that binds to our `todoModel` variable from our `TodoController`. This is binding the value of the input tag at any point in time to the `todoModel` object. Thirdly, we have set an `ng-click` attribute on the `add todo` button to ensure that any time it is clicked, a new `todo` item is added to our array. Finally, we have done an `ng-repeat` within the `UL` element to specify that we want all children of the `todo` array to be rendered with the `LI`.

With this, we have successfully completed the `todo` application and all that is left is to see it in action. We will be learning how to run this application we have just built-in different ways in the next chapter, so do follow up to learn how to get your app to run and test it live.

## Summary

In this chapter, we got to create our very first Ionic application using the Ionic blank application template. We had a look at what the Ionic workflow looks like and also got to see some of the files that make up the workflow. We then dived in and discussed about how we intended to build our to-do list application. We further went ahead and actually implemented the UI of our to-do list app based on a wireframe. We wrote some Angular code and wired it up to the user interface we created.

In the next chapter, we will learn different ways to run and test our application for the very first time with the Ionic CLI.



# 3

## Running Ionic Apps

In this chapter, we are going to learn how to test and run our Ionic application using various methods. We will start by learning to test our application using the simplest Ionic technique: by serving our app to the Chrome browser using the `ionic serve` command. We will then go ahead and use the Ionic view mobile app for iOS/Android to see how we can test our application on a mobile device. Lastly, we will learn to run and deploy our Ionic application to a mobile device using the traditional build system of the native SDKs of our respective platforms.

### Running our todo app

In the last chapter, we created our first Ionic application using the Ionic blank template. We worked on the application further, and made a to-do list app. We wrote some Angular code and had some initial exposure to some Ionic code. However, we did not get to see our application in action. There are many ways by which we can run an Ionic app, and the first technique we will be learning is the `ionic serve` technique.

### The `ionic serve` technique

The `ionic serve` technique is the simplest way to see your app in action. It requires no extra setup after the Ionic CLI, and only requires you to have a web browser. We are now going to test our `todo` application, which we created in the preceding chapter using the `ionic serve` technique. To test your application with this technique, simply open a new command-line window and follow the following steps.



### Browser choice

It is advisable that you use Google Chrome as your default browser. Google Chrome has some very powerful development tools and all exercises in this book expect that you have Google Chrome installed as your default browser. You can download a copy of Google Chrome by visiting this URL: <http://www.google.com/chrome>.

1. From your terminal, navigate to the root directory of your Ionic todo application.
2. Run the following command in your command-line window:

```
ionic serve
```



In case you are prompted to select an IP address, you can select any one from the list prompted and press *Enter* to initiate.

If you followed the steps correctly, you should see a browser window come up with your app running in it. You will also notice that the command-line window where you typed the command has some things going on within it.

With this, we have successfully served our application to the browser and can test our Ionic application like any other web application on Chrome. The great thing about this technique is the fact that no extra setup is required, and all you need is just Ionic CLI and the Chrome browser installed on your machine.

## Emulating with Chrome

Even though our application is served on the Chrome browser, it is fullscreen and is served like a normal fullscreen web app. This is not ideal for us, as our application is a mobile application. Luckily, Chrome has a neat emulation tool that lets you emulate your application as if it were running on a normal mobile phone.

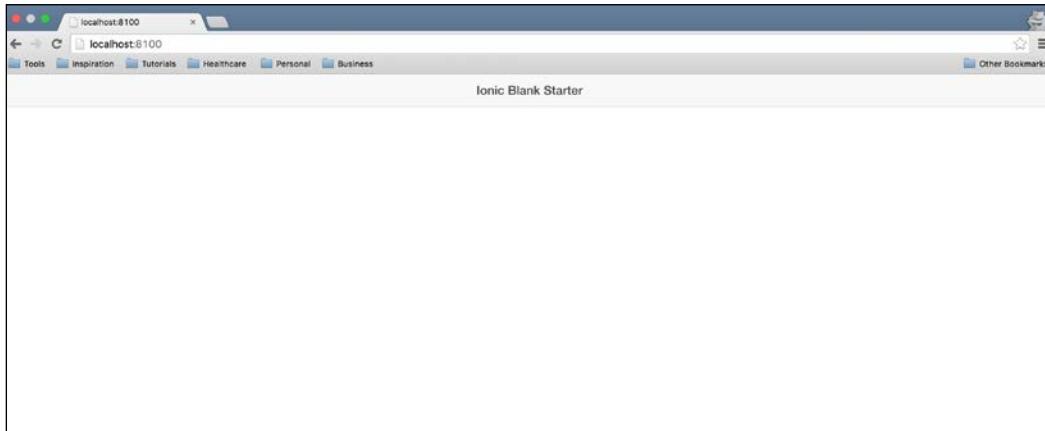
To use Chrome's emulation feature, follow the following steps.



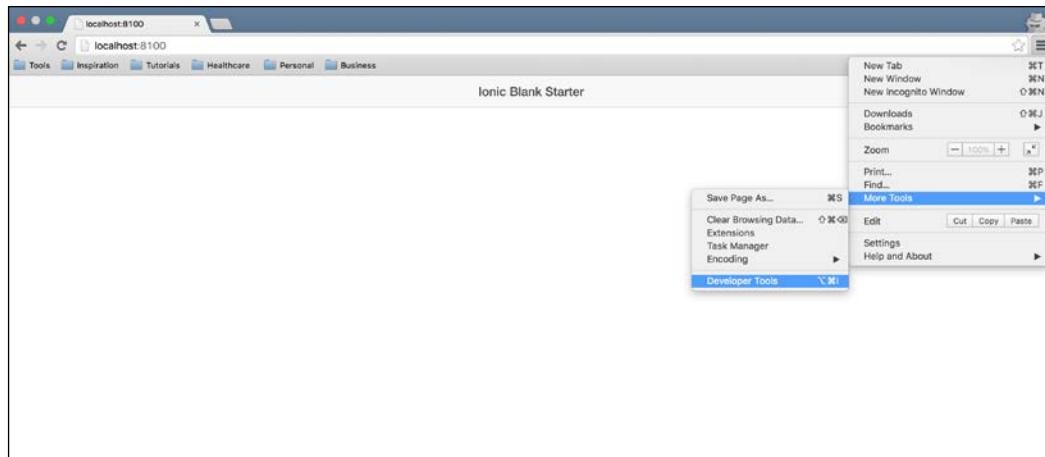
These steps assume that you already have your app served on the Chrome browser and you are currently on the tab that it is served on.



1. Click the Chrome menu icon, as shown in the following screenshot:

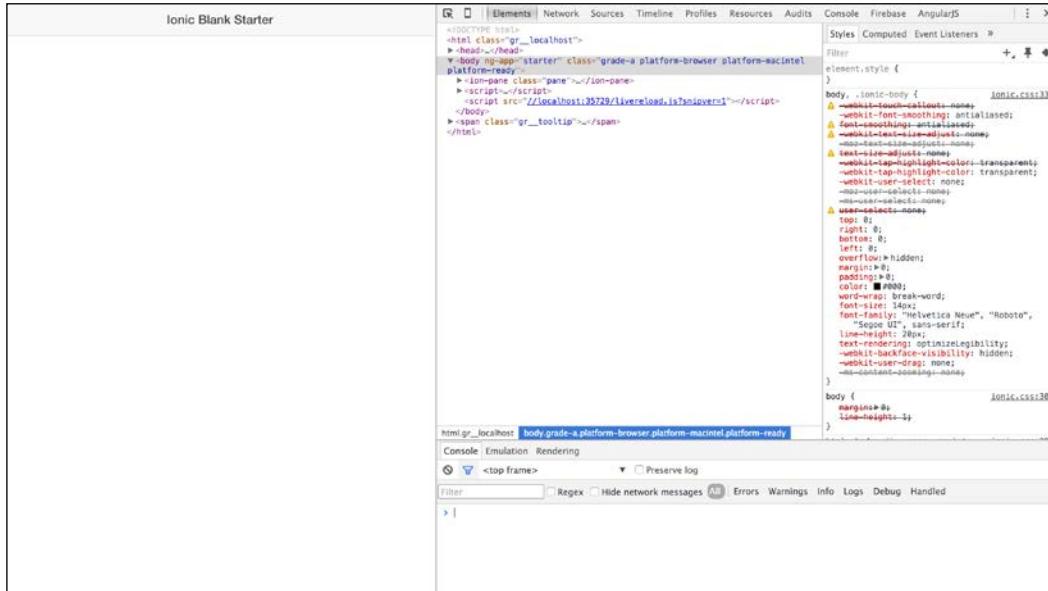


2. Scroll down to the **More Tools** options and select the **Developer Tools** option, as shown in the following screenshot:

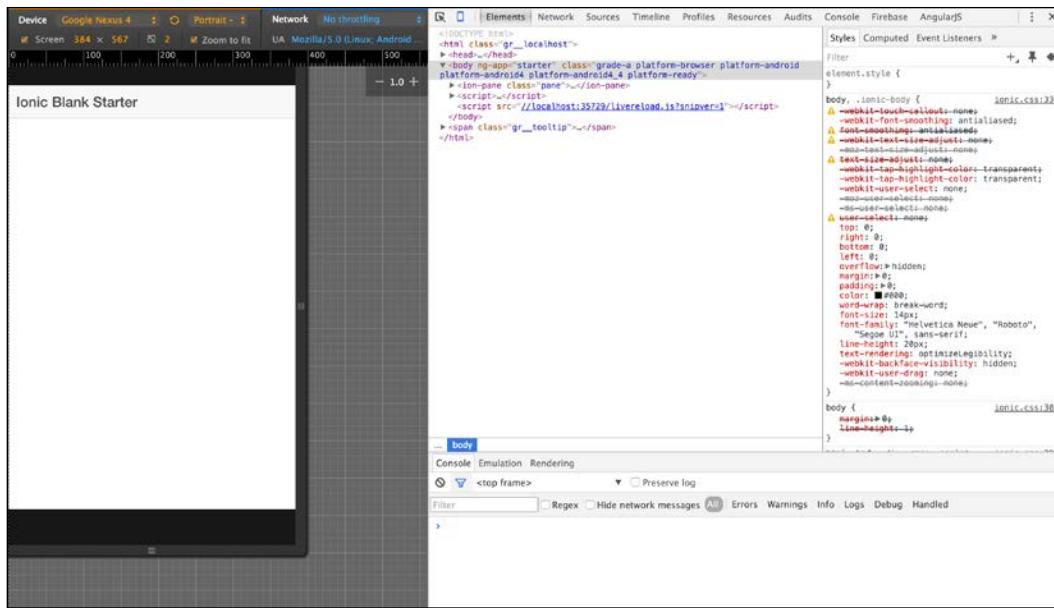


## Running Ionic Apps

3. Click the **Device Mode** toggle icon, as shown in the following screenshot:



This should bring up the Chrome emulator window with your app running on it. You might need to refresh the page for it to render the app correctly. If you have a look at the window, you will see a dropdown menu on the upper-left corner that has a list of devices that you can emulate. I normally recommend using the Nexus 5 for testing Android and the iPhone 6 for iOS. The reason for this is that the resolution of the Nexus 5 eclipses many of the Android phones available today so using it as a basis makes a lot of sense. The same goes for the iPhone 6 as well; since it is Apple's flagship device at the time of writing, it makes sense to use it for emulation.



You can fully interact with your app as if it were running in an emulator. You also have the full power of the Chrome developer tools to inspect elements and see how the code of your application is represented. Why don't you have a go with your app and try and add some to-do list items and see them populating.

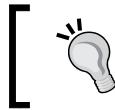
## Ionic serve labs

There is another flavor in the `ionic serve` technique that lets us see our app the way it looks on both iOS and Android simultaneously. This technique is called the Ionic labs technique.



This technique should only be used to view your app and is not intended to be used for debugging.

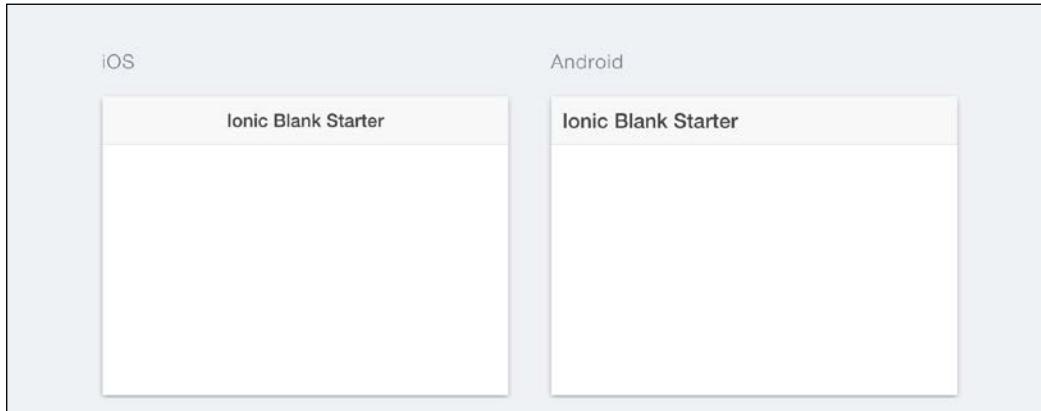
To view your app using the Ionic labs technique, simply follow the following steps.



If you are already viewing your app using the `ionic serve` technique, simply press `Q` to quit the current session or close the command-line window and open a new one.

1. Make sure you are in the root folder of your project.
2. Enter the following command in your terminal:

```
ionic serve --lab
```



Running this should bring up a new browser window the same way as it did the first time when we ran the `ionic serve` command; only that this time, you will see two emulations for your app, one for iOS and one for Android, as shown in the preceding screenshot. This is a really nice way to see your app running in action on both platforms simultaneously. Ionic has a term called **Continuum** which you will see in action in later chapters. This phenomenon refers to the fact that certain elements look different on different platforms. For example, tabs on iOS are normally placed on the bottom, while on Android, they are traditionally positioned on the top. Ionic offers us these features out of the box with a further way to override these behaviors. The `ionic serve` technique is a great way to see the features like the tab positioned differently on different platforms simultaneously.

## The Ionic view

Another technique to view an Ionic app is by using the Ionic view application. The Ionic view app is a mobile application created by Ionic with Ionic framework available on iOS and Android. The application is used to view any Ionic application you are developing and works hand-in-hand with the Ionic IO platform. The Ionic IO platform is a suite of tools that Ionic provides for some extra services like push notifications, analytics, and so on.

## Testing todo app with the Ionic view

In order to use the Ionic view app, you must have an iOS or Android device.

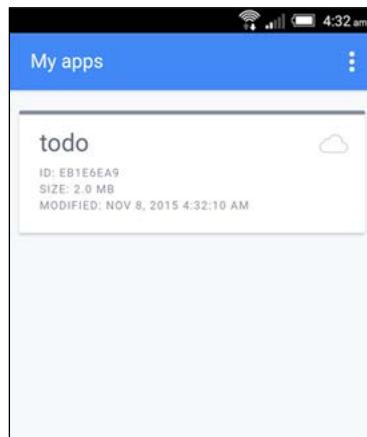
You must also possess an Ionic IO account. Navigate to <http://apps.ionic.io> to create your Ionic IO account. Go ahead and download the Ionic view app by visiting <http://view.ionic.io> on your mobile device and downloading the correct version for your mobile device.

In order to test our `todo` application, follow the following steps to test it with the Ionic view app:

1. Open a terminal window and navigate to the root folder of your `todo` application from Chapter 2.
2. Simply enter the following command on your terminal:  
`ionic upload`

This command will request the e-mail and password details of your Ionic IO account. Enter these details when prompted and if the app uploaded correctly, you should see a message saying `Successfully Uploaded (APP_ID)`, where `APP_ID` is an auto-generated identifier for your app.

Now you are ready to view the app on your mobile device. To do this, simply open your Ionic view app on your phone and login with the same Ionic IO account you uploaded your app to. You should see your application in a similar fashion to the following screenshot:



From here, you simply tap the `todo` app and a prompt will come up with a number of options. You should select the **download files** option. After this has finished, you can simply click the **View App** option. If you followed the instructions correctly, the `todo` app should replace your current view and you should see it running within the Ionic view app.

[  You can simply tap the screen with three fingers to go back to the Ionic view menu at any time. ]

The Ionic view is a good way to view your application, and is extremely useful when you want to share progress with your friends, clients, or your boss about an app. It has a feature that lets you share to people's e-mails and you can find these documented in the Ionic official documentation. You can also manage the apps you upload to Ionic view from within the app or online via the Ionic IO website at <http://apps.ionic.io>.

## Device

You can also test your Ionic application by running it on a physical device. To do this, however, you must have the native SDK for each platform installed on your computer. Let's take a brief look at how you can run an Ionic app on your device.

### Android

To run an Ionic app on a physical device, first you simply ensure that you have your Android device plugged in via USB. You also need to ensure that you have developer mode enabled in your computer with USB debugging on.

[  This step assumes that you have already set up the Android SDK on your computer and you also have Cordova and Ionic set up on your machine. ]

Ensure that you are in the root folder of your project in a terminal window and run the following command:

```
ionic run android
```

If you have everything set up correctly, this command will build your app and run it on the device plugged into the computer automatically.

## iOS

To run an Ionic app on an iOS device, first you need to ensure that you have the `ios-deploy` package installed.



You can only deploy your app to an iOS device using a Mac computer. This step also assumes that you have the iOS SDK set up correctly alongside X-Code on your Mac computer.

If you do not have the `ios-deploy` package installed, you can install it via NPM by running the following command:

```
npm install ios-deploy -g
```

Plug in your device to your Mac computer and ensure that it does not have the lock screen enabled. Simply run the following command to deploy your Ionic app to your device:

```
ionic run ios --device
```

This command should build and run your application automatically on your plugged iOS device.

## Summary

In this chapter, we learned the various ways to test and deploy our app. We started off by using the `ionic serve` command to deploy our app to the browser using Chrome. We then had a look at how we can also serve our application using Ionic labs. We then went ahead to use the Ionic view application to see how we can run our app on an iOS and Android device with the Ionic view app installed in it. Lastly, we touched on how we can actually run our Ionic application on a real Android or iOS device.

In the next chapter, we are going to dive into some more complex Ionic controls, and we will get to use Angular's `$http` service to see how we can make Ajax calls and retrieve data within our Ionic application.



# 4

## Ionic Components

In this chapter, we will be learning how to use some more complex Ionic components and controls. We will be creating a more advanced version of our to-do list application we created in *Chapter 2, To-Do List App*, using some more advanced built-in Ionic list components. We will call this more advanced to-do list application Bucket-List app. The idea behind this application is that it will allow us to enter all the interesting things we want to try in a lifetime. Therefore, we can enter the names of places we want to visit, the names of activities we want to do, and so on.

### Creating a new to-do list application

In *Chapter 2, To-Do List App*, we created a simple to-do list application with the Ionic blank template. We were able to get this application to work by allowing us to add items into our to-do list application. We will be creating a new to-do list application using the Ionic blank template for us to add our new, more advanced components to our brand new BucketList application. Let's go ahead and create this new blank application by following the following steps. We will be calling our new application Bucket-List in order to differentiate it from the one we created in *Chapter 2, To-Do List App*.

1. To create the Bucket-List app, fire up a terminal window on your computer and navigate to the Desktop folder of your computer by running the following command:  
`cd Desktop`
2. After navigating to the Desktop folder of your computer, go ahead and run the following command to create the Bucket-List application based on the Ionic blank template:  
`ionic start Bucket-List blank`

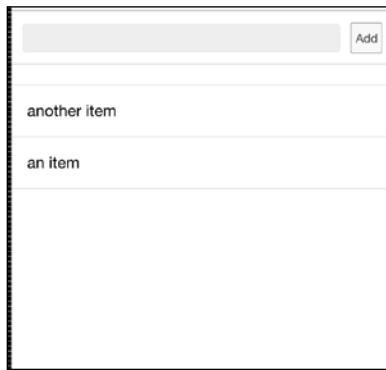
3. This command will create a new blank Ionic application called Bucket-List. When this command has finished running, navigate to the project of your application via the command line by running the following command:

```
cd Bucket-List
```

Now you have successfully completed the process of creating your Bucket-List application, and we can start developing the app by adding features to it.

## Overview of the Bucket-List app

To understand what we are trying to build, have a closer look at the following screenshot. We aim to achieve a final app that closely resembles what we have in the following screenshot:



## Breaking down the app

A good way to build Ionic apps is by building them in bits. For our Bucket-List application, we can start by first developing the user interface and then writing the code behind it to enable it to work. Our user interface will contain an input box to enter a new item into our bucket list. Secondly, we have to design the UI for the list of Bucket-List items.

## Designing the UI

Designing the UI involves two main implementations:

- Implementing the input box
- Implementing the `ion-list` element

We will have a look at each.

## Implementing the input box

The first thing we are going to implement is an input box. This input box is the form where the users of our app will enter an interesting item they wish to add in the Bucket-List application. This will be in the form of an HTML textarea input box with some Ionic CSS styles applied to it in order to give it a more mobile look and feel. There also will be a button next to the input box with the label **ADD**. This button will be what we tap after we have typed some text and want it to appear as a part of our list. Perform the following steps:

1. Open up the Bucket-List application you created earlier in your favorite text editor.
2. Now, open the `index.html` file that can be found in the `www` folder of your project. You will see a screen that closely resembles what we have in the following screenshot:

```
<!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">

    <!-- If using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">

    <ion-pane>
      <ion-header-bar class="bar-stable">
        <h1 class="title">Ionic Blank Starter</h1>
      </ion-header-bar>
      <ion-content>
        </ion-content>
    </ion-pane>
  </body>
</html>
```

You can see that this boilerplate code already contains some code for some custom Ionic elements just like we saw in our first application in *Chapter 2, To-Do List App* all of which are prefixed with `ion`. Pay close attention to the `<ion-content>` element! This element is the element that contains the bits and pieces of our application or the content area. It is in between this element that we are going to place all the markup for our Bucket-List application.

Let's start by placing the code for the input box of our application. I have provided the code for our input box in the following code block. You are to place this code within the `<ion-content>` element in your `index.html` file:

```
<div class="list">

  <div class="item item-input-inset">
    <label class="item-input-wrapper">
      <input type="text">
    </label>
    <button class="button button-small">
      Add
    </button>
  </div>

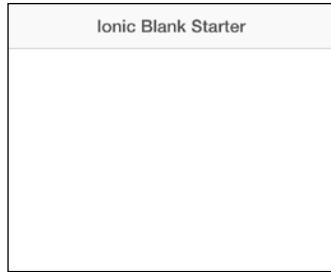
</div>
```

The preceding code is the HTML code that will display an input box and a button as we described earlier. If you pay close attention to the markup, you will see that some elements contain some classes. These classes are custom Ionic classes that are available as part of the Ionic CSS. The Ionic CSS comes with a lot of nifty classes and features, but for now just be aware of these classes and know that they are part of the Ionic CSS.

If you run your app in the browser using the `ionic serve` method, you should be able to see something that looks very similar to what I have in the following screenshot. Enter the following command in a terminal window to run your app using the `ionic serve` method. Make sure you run it from the root folder of your project:

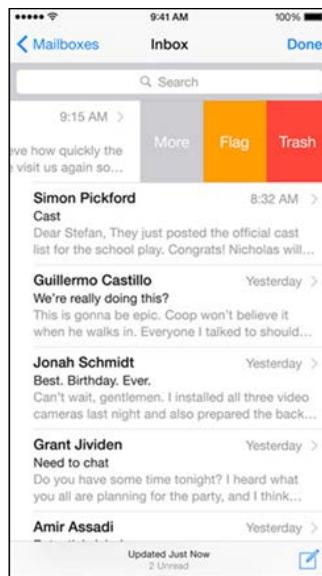
**ionic serve**

You should be able to see the input box with the button placed on its right-hand side.



## Implementing the ion-list application

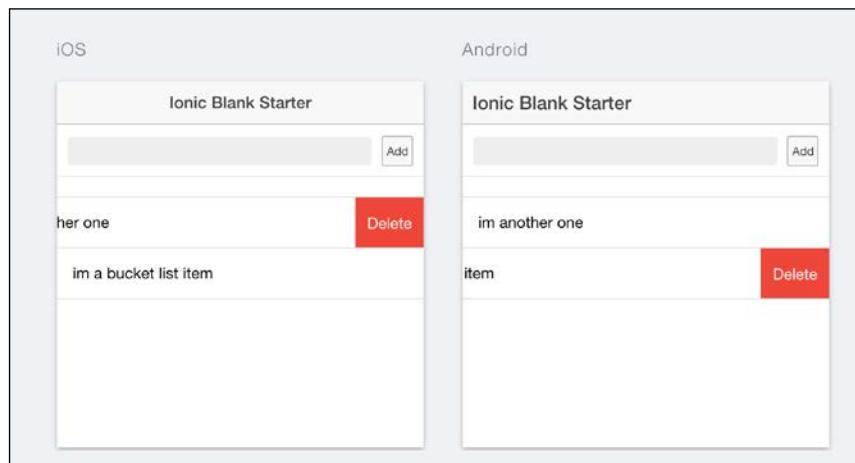
The next step of developing our Bucket-List application is to implement the ion-list application. We are going to use one of the built-in components called ion-list. The ion-list element is a component for creating and rendering lists. It has a lot of cool features that let us render complex lists that can have side options. Take a look at the following screenshot that shows the mail app from an iOS mobile device showing a list of features that we can implement using ion-list:



As seen above, one of the most obvious features we can implement with ion-list is the ability to show options when we swipe on a list item. It also has other features like the ability to delete items or rearrange them.

## Using ion-list for our Bucket-List app

For our Bucket-List application, we will be aiming to use the `<ion-list>` component to render every item we enter through the input box. In addition, we would want to be able to delete each item from the list by simply swiping from the left and thus revealing a delete button, which we can click. The following screenshot gives a sample graphic breakdown of what we aim to achieve and what items are involved:



## The ion-list component

The first thing we will do is implement the code for our `<ion-list>` component. The following code is the code for `<ion-list>`. You are to copy this code and place it just below the code for the input box you already implemented:

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

This is the top-level component needed to create our `<ion-list>` component. The `<ion-list>` component has some attributes that let us perform some more complex implementations. We will not be exploring these attributes but it is worth knowing that they do exist.

The next step is to implement the child item for our `<ion-list>` component. Each item in an `<ion-list>` component is called `<ion-item>`.

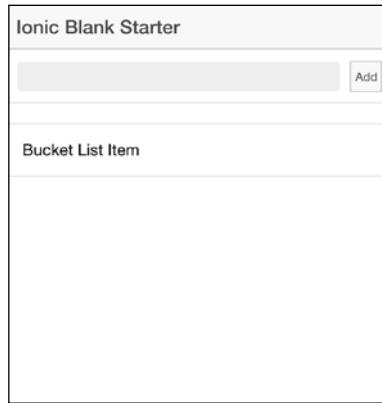
## ion-item

As briefly noted in the previous subsection about the `<ion-list>` component, each item in an `<ion-list>` is called an `<ion-item>`. Within an `<ion-item>`, we will put the code for what we want each list item to render. It is also the `<ion-item>` where we will use Angular's `ng-repeat` feature. For our application, we simply want to render the name of each Bucket-List item. This means that we can think of every Bucket-List item as an `<ion-item>`.

For now, we are just going to have some dummy text in our implementation showing how a Bucket-List item will prospectively look. The following block of code is the implementation of our `<ion-item>` representing a Bucket-List item in our app:

```
<ion-item>
<h2>Bucket List Item</h2>
</ion-item>
```

This is a sample representation of our `<ion-item>`. If you still have your app running in the browser via the `ionic serve` technique, you should be able to see the `<ion-item>` rendered just like in the following screenshot:



Now, with that implementation completed, there is one more thing we need to do in order to finish the implementation of the user interface of our Bucket-List app. The one thing remaining is the delete feature. Remember from our initial implementation plan that we want the user to be able to swipe each item in our list and have a **Delete** button revealed. Luckily for us, the `<ion-item>` component has a neat feature for this called the `<ion-option>`.

### ion-option-button

The `<ion-option-button>` component lives within an `<ion-item>` component as its child. Its sole purpose is to allow us to define buttons that we can reveal when the user of our app swipes from the right of each `<ion-item>` component just like in the original sample screenshot of our implementations. To get this implementation underway, copy the following code and paste it just before the closing tag of your `<ion-item>` component markup:

```
<ion-option-button class="button-assertive">
    Delete
</ion-option-button>
```

If you have a look at the preceding code, you can see that `<ion-option-button>` has a class attribute of `button-assertive`. This is also another Ionic class that is used to define a red button by default on Ionic buttons. Ionic has some built-in classes to easily add colors to elements. We will be discussing this later on in this book but for now just be aware of this feature.

By now, your final code for your `<ion-item>` component should look something similar to what I have in the following code block:

```
<ion-item>

<h2>Bucket List Item</h2>

<ion-option-button class="button-assertive">
    Delete
</ion-option-button>

</ion-item>
```

Your final code for your `<ion-content>` component should closely resemble what we have in the following code block:

```
<ion-content>
<div class="list">
<div class="item item-input-inset">
<label class="item-input-wrapper">
<input type="text">
</label>
<button class="button button-small">
    Add
</button>
</div>
```

```
</div>

<ion-list>
<ion-item>

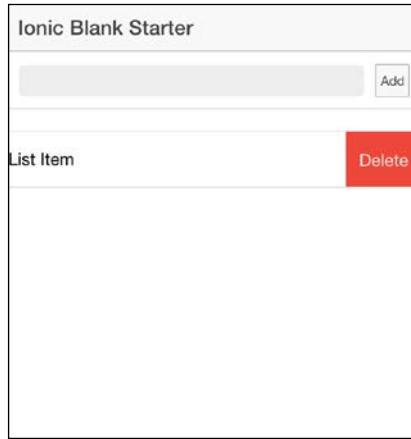
<h2>Bucket List Item</h2>

<ion-option-button class="button-assertive">
    Delete
</ion-option-button>

</ion-item>
</ion-list>

</ion-content>
```

With this, we have completed the implementation of the user interface of our Bucket-List application using the `<ion-list>` component. Provided you still have your app running in the browser via the `ionic serve` technique, go and try to swipe the sample list item in your app from the left-hand side. You should be able to see a **Delete** button when you do this. See the following screenshot for guidance:



With this step completed, we are finished with the user interface of our application completely. Now, it is time we start to wire up the app by focusing on the Angular code that we will be writing to ensure our application works the way we want it to.

## Writing the Angular code for our Bucket-List app

Before we begin, let's recap what behavior we want to implement in order for our application to work the way we want it to.

### Coding our input box

The first thing we want is to be able to enter some text into our input box later. After we enter the text, we want to click the **Add** button and have this text entered into an array that holds all our Bucket-List items. To begin this first, we create our Angular controller that will hold all the logic for our app.

### Creating the controller

Open to the `app.js` file of your application in your favorite IDE. This file can be found in the `js` folder, which is found in the `www` folder of your app.

`www/js/app.js`

There should already be a folder called `starter` with code similar to that which I have in the following code block:

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

.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();
    }
  });
})
```

To begin, we will first start by creating a controller called `BucketListController` just after where we declared our module. If you have done this correctly, you should have code that closely resembles the following:

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

.controller('BucketListController', function ($scope) {

})

.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();
        }
    });
})
```

With this done, we have now completed the process of creating our controller called `BucketListController`. This controller is where all the logic for our app will live.

## Creating the input box model

We are going to need to create a model that will be bound to our input box. This model will be in the form of an object, and it will hold the data that will be represented by the text we enter in our input box. We will call this model `bucketListItem`. To create this model, simply enter the following code within `BucketListController` that you just created in the previous step:

```
$scope.bucketListItem = {
    title: ''
};
```

The preceding code is initializing the model for our `bucketListItem` model. This model has a property called `title` that will hold the text of what we type in the input box at every point in time.

## Creating an array for the Bucket-List items

The aim of our app is to have a list of the Bucket-List items. These items, as we enter them in our input box, will need to be stored in an array. We are going to create this array, and we will simply call it `bucketListItems`. This array is what we will use in Angular's `ng-repeat` attribute to iterate and render in our view. To create this array for our Bucket-List items, simply attach an array called `bucketListItems` to the `$scope` variable of your controller. The following code illustrates this step:

```
$scope.bucketListItems = [] ;
```

## Implementing code for the Add button

The final step to ensure that we are able to add items to our Bucket-List app with the input is by writing the code in the form of a function for the **Add** button. This button is responsible for two things. Firstly, it will ensure that the current text in our input box is added as an entry to the output box array of the Bucket-List items. Secondly, it will also clear up the model to ensure that after we click the button, the input box is cleared up for the next item.

The following code represents the implementation for our **Add** button:

```
$scope.addBucketListItem = function () {
    //Add Current Bucket List Item To The Front Of Our Bucket List
    //Items Array
    $scope.bucketListItems.unshift($scope.bucketListItem);
    //Clear Current Bucket List For Next Entry
    $scope.bucketListItem = {
        title: ''
    };
};
```

From the preceding code you can see that we have created a function called `addBucketListItem`, and attached it to the `$scope` variable of our controller so it is available to our view. Within our function, we first add the current value of the `bucketListItem` variable to our `bucketListItems` array. Secondly, we clear up the `bucketListItem` variable to ensure it is cleared for the next entry.

Now, you can go ahead and implement the preceding code within your controller.

## Implementing the Delete button

The last piece of our code is to implement the **Delete** button of our `<ion-option-button>`. Remember that we want this button to be able to delete the item which it belongs to. The following code shows the implementation of the **Delete** button:

```
$scope.deleteBucketListItem = function (index) {
  $scope.bucketListItems.splice(index, 1);
};
```

The preceding code simply shows how we have created a function called `deleteBucketListItem`. This function takes in the index of the current item to be deleted as a parameter. It then uses this index to remove the corresponding element that is placed in that index from the `bucketListItems` array, which holds all our Bucket-List items.

With that complete, we have pretty much finished the code aspect of our application. Your final controller should look similar to what I have in the following code block:

```
controller('BucketListController', function ($scope) {
  $scope.bucketListItem = {
    title : ''
  };

  $scope.bucketListItems = [];

  $scope.addBucketListItem = function () {
    //Add Current Bucket List Item To The Front Of Our Bucket
    List Items Array
    $scope.bucketListItems.unshift($scope.bucketListItem);
    //Clear Current Bucket List For Next Entry
    $scope.bucketListItem = {
      title: ''
    };
  };

  $scope.deleteBucketListItem = function (index) {
    $scope.bucketListItems.splice(index, 1);
  };
})
```

Now, before we go ahead and test your application, we have one last step to complete. We need to wire up all the code we have just created with the UI we implemented earlier so that they can work together.

## Wire it all up

With our controller ready, now we have to go ahead and wire all the code to the UI so that they can work together in harmony.

### Binding the controller

The first thing we need to do is to wire up the controller we created. The simple and easiest way to do this is by simply using Angular's `ng-controller` attribute directive to specify our controller. In our case, we will be wiring the controller on `<ion-content>` of our app. Once again, open up the `index.html` file of your application. Find the opening `<ion-content>` tag of the page and specify an `ng-controller` attribute with the name of your controller.

Your code should closely resemble the following code:

```
<ion-content ng-controller="BucketListController">
```

This code is simply telling Angular that we wish to use `BucketListController` within the scope of this `<ion-content>` element. This means that all the methods and properties scoped within this controller are now available to the `<ion-content>` element and all its descendant elements.

### Binding the input box model

The next step is to ensure that the `bucketListItem` variable we created in our controller is data bound to our input box in the view. Angular also has a simple but great attribute directive for this called `ng-model`. We simply provide `ng-model` with a value that matches an object or variable that we want to data bind to. In our case, we want to data bind to the `title` property of our `bucketListItem` variable from our controller. Again, I have provided the following code for your convenience:

```
<input type="text" ng-model="bucketListItem.title">
```

The preceding piece of code we just added tells Angular to bind this variable to this input box. Therefore, anytime the value of the input changes from the view, we have the same value in our controller and vice versa.

### Wiring up the Add button

The **Add** button is next in line for our implementation. For this button, we simply need to tell it to run our `addBucketListItem` function every time it is clicked. Once again, Angular has a helper directive for this called the `ng-click` directive. The `ng-click` directive is like the classic Java `onClick` event listener and you provide it with a function that you want to run every time the wired element is clicked. The following code demonstrates how we can wire up our **Add** button with the `ng-click` directive:

```
<button class="button button-small" ng-
click="addBucketListItem() ">
    Add
</button>
```

The preceding code implementation simply ensures that when the **Add** button is clicked, the `addBucketListener` function will run with its expected behavior.

## Binding ion-item

The last part of our wiring up will be to wire our `bucketListItems` array to our `Ion-Item` elements, and also bind the `ion-option-button` element to our `deleteBucketListItem()` function.

## Using ng-repeat to render the list

Right now we have a sample implementation that has one hardcoded `ion-item`. However, we will want a more dynamic solution where we automatically render the items within the `bucketListItems` array each as an `ion-item`. For this implementation, we are going to use one of the most important Angular features in the form of `ng-repeat`. The `ng-repeat` angular directive lets us dynamically repeat an array.

Right now, you have a code that looks similar to the following:

```
<ion-item>

    <h2>Bucket List Item</h2>

    <ion-option-button class="button-assertive">
        Delete
    </ion-option-button>

</ion-item>
```

We are going to change this implementation to use the `ng-repeat` directive of Angular. The following code shows you how this is achieved:

```
<ion-item ng-repeat="item in bucketListItems">

    <h2>{{item.title}}</h2>

    <ion-option-button class="button-assertive">
        Delete
    </ion-option-button>

</ion-item>
```

The preceding code now uses Angular's `ng-repeat` attribute. This code tells Angular to repeat the `bucketListItems` array and also binds the title of each item to an HTML `<h2>` element.

## Wiring up the `ion-option-button` element

The `ion-option-button` element is still untouched and will do nothing if we don't tell it to do so. All we need to do for this element is to provide it with a function we want to be executed when it is clicked, like we did with the **Add** button. For this, we will be using the `ng-click` directive again, but this time, we will point it to the `deleteBucketListItem()` function from our controller. The following code shows just how we can achieve that:

```
<ion-option-button class="button-assertive" ng-
click="deleteBucketListItem($index)">
    Delete
</ion-option-button>
```

From the preceding code, you will notice one alien thing, `$index` being specifically passed as a parameter for our `deleteBucketListItem` function. This variable is a magic variable that the `ng-repeat` directive of Angular exposes to us. It represents the index of the current element being rendered by `ng-repeat`. With this index, we can learn what particular element should be deleted from our array of bucket list items, and delete the correct one.

The final `<ion-content>` in your `index.html` file should closely resemble what I have in the following code block:

```
<ion-content ng-controller="BucketListController">

    <div class="list">
        <div class="item item-input-inset">
            <label class="item-input-wrapper">
                <input type="text" ng-model="bucketListItem.title" />
            </label>
            <button class="button button-small" ng-
click="addBucketListItem()">
                Add
            </button>
        </div>
    </div>

    <ion-list>

        <ion-item ng-repeat="item in bucketListItems">
```

```
<h2>{{item.title}}</h2>

<ion-option-button class="button-assertive" ng-
click="deleteBucketListItem($index)">
    Delete
</ion-option-button>

</ion-item>

</ion-list>

</ion-content>
```

## Testing our Bucket-List app

We have completed the implementation of our application, and now it is time for us to see it in action. Ensure you have your app running in the browser via the `ionic serve` technique, and test it. Try entering some things into your Bucket-List app such as skydiving, jet-skiing, and so on. You should see that every time you enter an item and click **Add**, the item will appear in the list and the input box will clear up ready for your next input. Also, make sure you test the delete option by swiping an item from the left to reveal the **Delete** button, and clicking it to see the item disappear.

## Summary

In this chapter, we focused on creating our Bucket-List application from scratch using the Ionic blank template. We also learned to use the `<ion-list>` component of Ionic and its child elements. We wrote some Angular code to wire everything up and got it running. The `<ion-list>` component is a very powerful component, and although the task of this chapter might appear a bit more complex than the previous ones, there are still some more powerful features that the `<ion-list>` component lets us do. For more information about `<ion-list>`, visit the official documentation of `<ion-list>` from the provided links in the appendix of this book to learn even more complex features.

In the next chapter, we will be learning some very exciting stuff about creating side menu applications with Ionic. We will also build ourselves a tourist application and work with the AJAX calls for the very first time using Angular's `$HTTP` service.



# 5

## The London Tourist App

In the previous chapter, we created an application called the Bucket-List application that enabled us to create a list of interesting things we wanted to do in our lifetime. In this chapter, we will create a new application called "The London Tourist" application. It is an application that will display a list of top tourist attractions in the city of London in England. We will build this application with a new type of Ionic template called the side menu template. We will also be using the Angular `$http` service to query our data via Ajax.

### Introduction to the London Tourist App

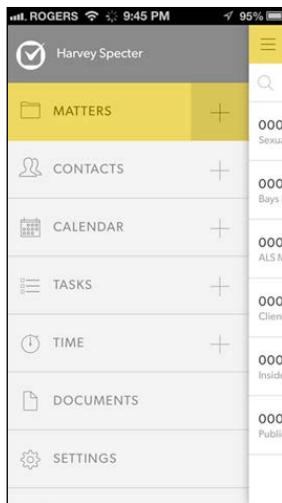
London is the largest city in England and it is a city that is well known to attract tourists around the world. The city is very urban but it has many historical and non-historical tourist attractions. With this large number of attractions, it can be difficult to pick out the best places to go. This is the entire idea behind the London Tourist App as it will provide users with five handpicked destinations that tourists visiting London can actually visit. These destinations will be stored in a JSON file in our project that we will be querying via AJAX and populating.

### Creating the London Tourist app

To begin the process of creating our app, we are going to start by creating a brand new Ionic application. So far in this book, we have learned to create a new Ionic application using the blank template. For the application we are about to build, we are going to use a new type of Ionic template to create the application. We are going to be using the side menu template to create our London Tourist app.

## The side menu app design

You might not be familiar with what the side menu template looks like. In fact, the side menu design for mobile applications is very common in mobile app development. It involves having the ability to slide from the left or right edges of a mobile application to reveal more options, normally more menu options:



The side menu design technique is one that is used in a lot of contexts, both on mobile and on the web. Normally, you will see an icon positioned either on the far upper-right or upper-left, indicating that you can swipe or click that icon to reveal the extra menu options. This icon is normally referred to as the hamburger menu icon.

The Ionic framework actually comes built-in with a side menu template that automatically creates a side menu application for us with some useful boilerplate code. We will be using this template to create our London Tourist Application.

## Using the Ionic side menu template

To begin developing our London Tourist Application, we will begin by using the Ionic CLI to create the app. You can do this by running the following command from a terminal window:



We will shorten the name of our app from London Tourist App to LTA to make it easier to type.

```
ionic start LTA sidemenu
```

This command will create a new Ionic application called LTA using the default Ionic side menu template.

## Seeing the LTA side menu app in action

As soon as your LTA app is created, you can simply change your directory into the app from the terminal and run it on your computer using the `ionic serve` technique. You can do this by running the following commands:

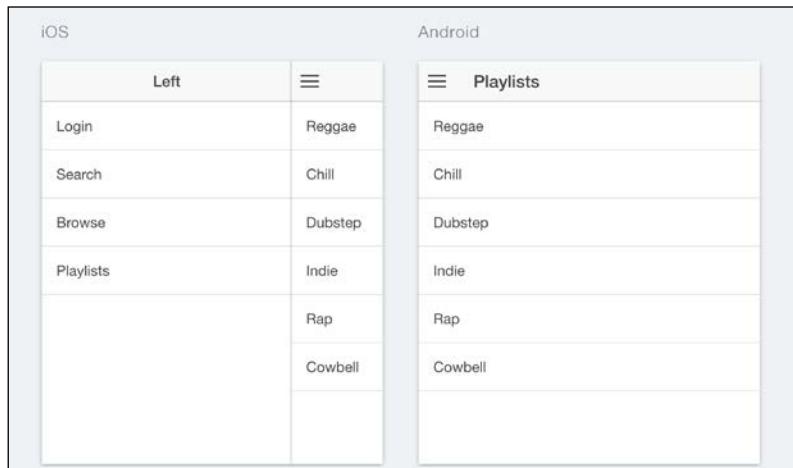
- Change directory to app:  
`cd LTA`
- Run using the `ionic serve` technique:  
`ionic serve`



Remember to use Chrome and emulate to a device of your choice with the Chrome emulation tools as taught in previous chapters.



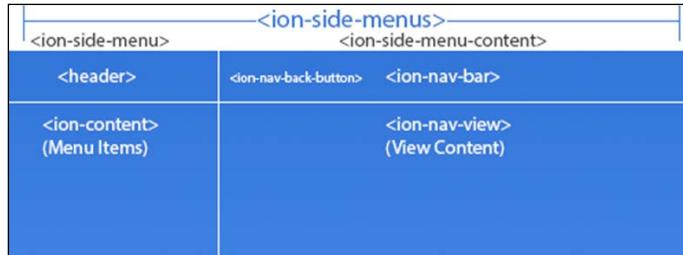
You should see a screen that looks something like the following screenshot:



As you can see from the preceding screenshot, the side menu app we have just created contains some pre-rendered content.

## Exploring the LTA side menu app's code

Now, we are going to have a look at the code of the LTA app based on the side menu template:



Now, I will require you to fire up the LTA project you have just created in your favorite IDE. The first thing you will notice is the folder structure that you are already used to from previous chapters.

### The index.html file

Now, focus on the www folder and open the `index.html` file. A look through this file should show you something similar to what we have in the following screenshot:

```
<!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">

    <!-- If using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
  </head>

  <body ng-app="starter">
    <ion-nav-view></ion-nav-view>
  </body>
</html>
```

[  To get to this file from the root folder, navigate to [www/index.html](#). ]

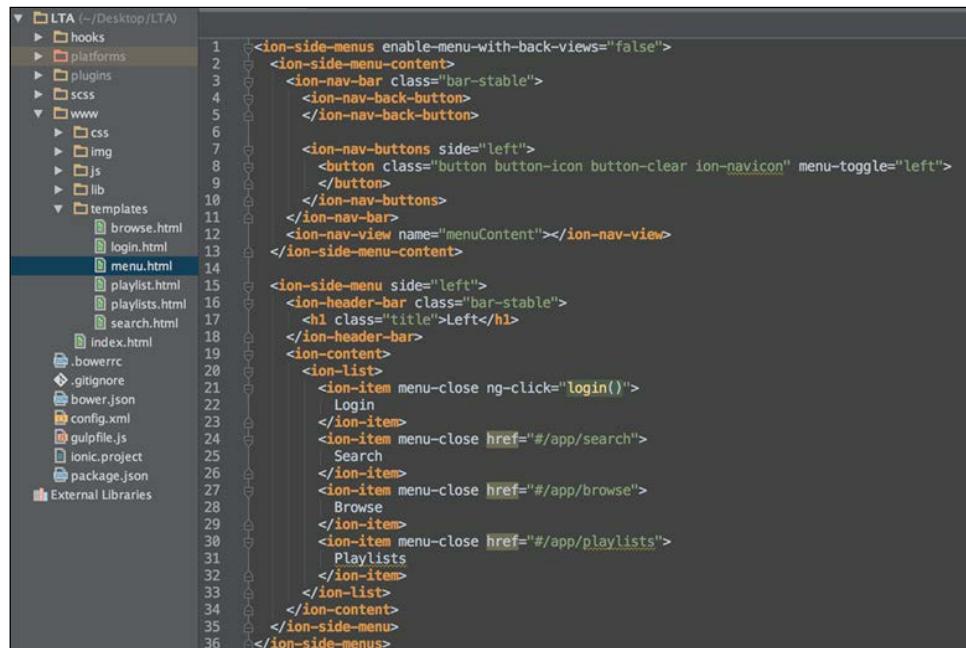
A look at this file will show you some things we have seen from previous projects in this book. For example, you can see some CSS and JS references to the Ionic styles and script files respectively. You will also see from the `body` tag that a generated Angular module called `starter` is being instantiated using the `ng-app` attribute.

Pay close attention to the `<ion-nav-view>` tags within the `<body>` tags. This is an Ionic element that is used to specify the view to which the entire app is injected into. It is similar to the `ng-view` feature of Angular but has more powerful features. It also automatically handles navigation for us within our Ionic app. You need not pay a great deal of attention to this part of the code anymore but just have it at the back of your mind that `<ion-nav-view>` is where all content gets injected in, and acts like a wrapper for our app's content.

## The menu.html file

The next file we are going to explore is the `menu.html` file. This file is probably the most important file at this moment as it contains most of the generated code for the side menu parts of our app. To have a look at this file, navigate to the `menu.html` file which can be found by navigating into the folder called `templates` under the `www` folder. Here is the path: `www/templates/menu.html`.

If you have successfully done this, you should see a file that closely resembles what we have in the following screenshot:



```

1 <ion-side-menus enable-menu-with-back-views="false">
2   <ion-side-menu-content>
3     <ion-navbar class="bar-stable">
4       <ion-nav-back-button>
5     </ion-nav-back-button>
6
7     <ion-nav-buttons side="left">
8       <button class="button button-icon button-clear ion-navicon" menu-toggle="left">
9         </button>
10      </ion-nav-buttons>
11    </ion-navbar>
12    <ion-nav-view name="menuContent"></ion-nav-view>
13  </ion-side-menu-content>
14
15  <ion-side-menu side="left">
16    <ion-header-bar class="bar-stable">
17      <h1 class="title">Left</h1>
18    </ion-header-bar>
19    <ion-content>
20      <ion-list>
21        <ion-item menu-close ng-click="login()">
22          Login
23        </ion-item>
24        <ion-item menu-close href="#/app/search">
25          Search
26        </ion-item>
27        <ion-item menu-close href="#/app/browse">
28          Browse
29        </ion-item>
30        <ion-item menu-close href="#/app/playlists">
31          Playlists
32        </ion-item>
33      </ion-list>
34    </ion-content>
35  </ion-side-menu>
36 </ion-side-menus>

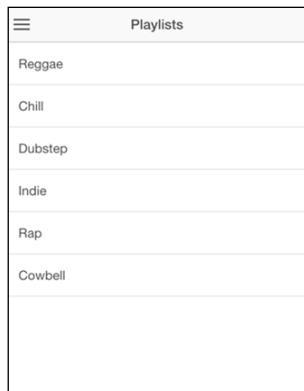
```

## The <ion-side-menus>element

The first thing you should pay attention to is the `<ion-side-menus>` element. Think of this element as a container for any side menu application. It allows us to specify what the main content area will be via the `<ion-side-menu-content>` element and also allows us to specify the side menus via the `<ion-side-menu>` elements. There can be more than one side menu specified within the `<ion-side-menus>` elements and we have the ability to specify whether the menu is placed on the left, right, or even both. There are a lot of cool and powerful controls that the `<ion-side-menus>` elements let us utilize to control its containing items. For now, we will focus on trying to learn more about the child elements that are needed to work with the `<ion-side-menus>` elements. These are the `<ion-side-menu-content>` and `<ion-side-menu>` elements.

## The <ion-side-menu-content>element

This element is what houses the main content area or the visible part of the app:



In the preceding screenshot, which is a view of our LTA app, the part you see is a representative of `<ion-side-menu-content>`. Let's have a closer look at the code of `<ion-side-menu-content>` to see how it actually works in detail:

```
2  <ion-side-menu-content>
3    <ion-nav-bar class="bar-stable">
4      <ion-nav-back-button>
5        </ion-nav-back-button>
6
7      <ion-nav-buttons side="left">
8        <button class="button button-icon button-clear ion-navicon" menu-toggle="left">
9          </button>
10     </ion-nav-buttons>
11   </ion-nav-bar>
12   <ion-nav-view name="menuContent"></ion-nav-view>
13 </ion-side-menu-content>
```

Within `<ion-side-menu-content>`, you can see two direct child elements.

Firstly, you can see the `<ion-nav-bar>` element which is used to build the navigation buttons of the main content area with the `<ion-nav-buttons>` element as its child element. For example, within these `<ion-nav-buttons>` elements, you can see a navigation button on line 8-9, which has a `menu-toggle` attribute of value `left`. This is simply saying that when this button is tapped, the left-sided side menu should be triggered. Remember that there can be up to two side menus with one being on the left and one being on the right in a side menu app.

The second direct child element is the `<ion-nav-view>` element on line 12 from the preceding screenshot. We talked about this same element earlier when we had a look at the `index.html` file. This element is a placeholder for where the actual content is injected. This particular `<ion-nav-view>` element has a `name` attribute with the value of `menuContent`. This attribute is important as it is used like a value to uniquely identify `<ion-nav-view>`.

With all that said, we have now lightly touched on the `<ion-side-menu-content>` element and its main functions. Always think of this element as the element that houses the main content area of your side menu application.

## The `<ion-side-menu>` element

The `<ion-side-menu>` element is an element that we use to specify the side menu of our app. Just like the `<ion-side-menu-content>` element, it lives as a direct child of the `<ion-side-menus>` element. There can be up to two `<ion-side-menu>` elements within the `<ion-side-menus>` element, with only one being on each side. Let's have a closer look at the code of `<ion-side-menu>` of our LTA application.

```
<ion-side-menu side="left">
  <ion-header-bar class="bar-stable">
    <h1 class="title">Left</h1>
  </ion-header-bar>
  <ion-content>
    <ion-list>
      <ion-item menu-close ng-click="login()">
        Login
      </ion-item>
      <ion-item menu-close href="#/app/search">
        Search
      </ion-item>
      <ion-item menu-close href="#/app/browse">
        Browse
      </ion-item>
      <ion-item menu-close href="#/app/playlists">
        Playlists
      </ion-item>
    </ion-list>
  </ion-content>
</ion-side-menu>
```

The preceding screenshot is from our `menu.html` file, and it showcases the code of `<ionic-side-menu>` from our LTA application. If you look at it closely, you will notice that opening tag of our `<ion-side-menu>` element has a `side` attribute with value `left`. This is basically saying that we want this particular side menu to be on the left-hand side. Remember that we can have up to two side menus in our app, and one can be positioned on the left and another on the right, but two side menus cannot be positioned on the same side. We can also see that this `<ion-side-menu>` has two direct child elements. These child elements are `<ion-header-bar>` and `<ion-content>`. `<ion-header-bar>` is an element used to construct the header of a side menu. If you have a look at the following screenshot of our side menu, you should see a representation of it:

Left	≡
Login	Reggae
Search	Chill
Browse	Dubstep
Playlists	Indie
	Rap
	Cowbell

From the preceding screenshot, you can see the header with the title `LEFT` as reflected in the code as an `<h1>` element.

The second child element we can see from the code is the `<ion-content>` element. Think of this element as what houses the content area below the header of the side menu. Basically, this is anything below the header. `<ion-content>` could contain any HTML code we want but in this case, it contains `<ion-list>` which is something that we used to build our Bucket-List application from *Chapter 4, Ionic Components*. You can also see a reflection of this code on the screenshot from when we ran our application.

With that said, you can see that we have successfully had a brief look at what the `<ion-side-menu>` element entails and how the side menu template of Ionic functions. The next step is for us to actually go ahead and build our LTA application in full scale now.

## Developing the LTA application

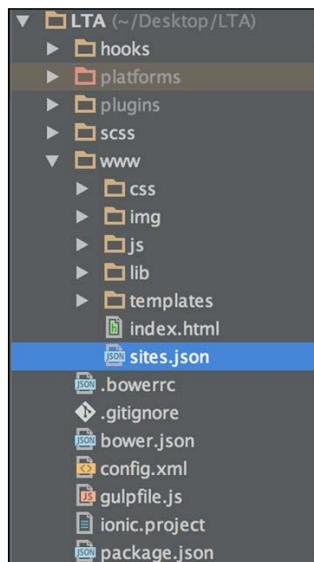
We are now equipped with the know-how on how to code our LTA side menu based application. Remember that the idea behind our application is to have some of our favorite tourist destinations listed in our app. In a normal scenario, we would query this data from a real API. But for the sake of simplicity, we will mimic this API request by making a request to a local JSON file that would act like a real database with the information we need.

### The local JSON database

As discussed earlier, we are going to create a JSON file that will act like a real-life API containing our destinations. This local file will contain five top tourist destinations in London. The first thing we will need to do is to create this file.

### Creating the local JSON database file

If you do not have your LTA application open, make sure you open it in your favorite IDE. Now, go ahead and create a new JSON file called `sites.json` within the `www` folder of your project. Make sure you name the file as the `.json` extension in order for it to be parsed as a JSON file. Your directory structure should look similar to what is shown in the following screenshot:



With that done, you have successfully created your local JSON file representing your database for your tourist sites.

## Populating the JSON file

Now we are going to populate the JSON file with some data. This data will be the data of five top tourist attractions in the city of London. The following is a JSON array that represents the content of our local JSON database. You should copy all the content of the following piece of code into your `sites.json` file:

```
[  
  {  
    "id": "1",  
    "name": "London Eye",  
    "description": "Shows you a great view of the city"  
  },  
  {  
    "id": "2",  
    "name": "The Shard",  
    "description": "Highest building in London"  
  },  
  {  
    "id": "3",  
    "name": "Oxford Circus",  
    "description": "The place to shop in London"  
  },  
  {  
    "id": "4",  
    "name": "Buckingham Palace",  
    "description": "The Queen lives here"  
  }  
]
```

The preceding piece of code is a JSON array that represents four top destinations in London as JSON objects. Each object representing a site has three properties. These properties are:

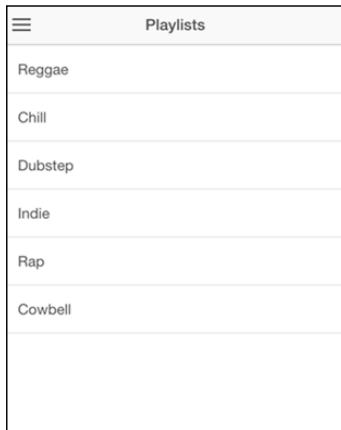
- **ID:** A unique identifier for the site.
- **Name:** The name of the Tourist site.
- **Description:** Some small information about the site.

By now, we have successfully completed the creation of our JSON local database. The next step is to see how we can actually render these items and query the database.

## Designing the view

We have created our app and we have the data for the application. Before we query data, we first need to design how the data will look when rendered. For this very task, we will call on an old friend of ours in the face of `<ion-list>`. We will use `<ion-list>` to render a list of tourist attractions from our JSON database.

Currently if we run our application, the first page we see is the playlist application, as shown in the following screenshot:



This is because by default the page is specified in the `app.js` file by Angular as the root page of our app. We will keep things simple and change the contents of this playlist page and design the view of our LTA application on it. From your LTA project folder, navigate into the `www` folder and look into the `templates` folder. Within the `templates` folder, there is a `playlists.html` file. This is the file that contains the code for our playlist page shown in the preceding screenshot. Open this file and you should see some code that closely resembles what we have in the following screenshot:

```
<ion-view view-title="Playlists">
<ion-content>
  <ion-list>
    <ion-item ng-repeat="playlist in playlists" href="#/app/playlists/{{playlist.id}}">
      {{playlist.title}}
    </ion-item>
  </ion-list>
</ion-content>
</ion-view>
```



The `playlists.html` file from the root folder of your LTA project will have a path `www/templates/playlists.html`.

The first thing we will want to do here is to change the name of the title of our view. Currently, the view as seen from the screenshots previously, has a title `Playlists`. This is specified by the `view-title` attribute of the opening `<ion-view>` element. This `view-title` attribute currently has a value `Playlists`. Change this to `London Sites`. This is to ensure that the title reflects the mission of our app, which is to show the top London tourist sites.

The second thing we need to do is to edit the code for `<ion-list>`. Replace the `<ion-list>` code with the one provided in the following code block:

```
<ion-list>
<ion-item ng-repeat="site in sites">
  {{site.name}}
</ion-item>
</ion-list>
```

If you have done this correctly, your code should now closely resemble what we have in the following screenshot:



A screenshot of an Ionic code editor showing the following code block:

```
<ion-view view-title="London Sites">
  <ion-content>
    <ion-list>
      <ion-item ng-repeat="site in sites">
        {{site.name}}
      </ion-item>
    </ion-list>
  </ion-content>
</ion-view>
```

With this done, we have now completed the process of designing our UI. The next step is to go ahead and wire up our data to our view.

## Wiring up the data

Earlier, we created a `sites.json` file that represented our database. We will be making a real Ajax call to this file in order to retrieve its data and serve it within our app. The thing we need to do to achieve this is firstly to write the code to retrieve the data.

## Retrieving the data with the `$http` service

To retrieve the data, we will need to make an Ajax call to the `sites.json` file. For this, Angular has a great service called the `$http` service. This is a service that provides us with functionality to make Ajax calls to local and remote resources via Ajax. To begin using the `$http` service to write our code, we first need to go to the controller associated with our view. By default, when you create an Ionic app based on the side menu template, there is a controller attached to the views. To find out which controller is attached to our `playlist.html` file, we need to look at the `app.js` file of our app to discover this.

You can find the `app.js` file by navigating to the `www` folder of your project and looking into the `js` folder within it. You should see the `app.js` file. Open it. After you open this `app.js` file, look thorough the part where you have code that looks closely to what we have in the following screenshot:

```
.state('app.playlists', {
  url: '/playlists',
  views: {
    'menuContent': {
      templateUrl: 'templates/playlists.html',
      controller: 'PlaylistsCtrl'
    }
  }
})
```

The code from the preceding screenshot represents the state definition of the `playlist.html` file. Pay close attention to the part of the code from the preceding screenshot where the controller is defined and you will see that the controller specified there is called `PlaylistsCtrl`. This is the name of the Angular controller that our `playlist.html` file is wired with.

The next step is to go to this `PlaylistsCtrl` controller and write the code to retrieve our data. By default, the controllers are contained in the `controller.js` file that can be found in the same `js` folder as our `app.js` file.



Open the `controller.js` file and look for a stub of code that closely resembles what I have in the following screenshot:

```
.controller('PlaylistsCtrl', function($scope) {
  $scope.playlists = [
    { title: 'Reggae', id: 1 },
    { title: 'Chill', id: 2 },
    { title: 'Dubstep', id: 3 },
    { title: 'Indie', id: 4 },
    { title: 'Rap', id: 5 },
    { title: 'Cowbell', id: 6 }
  ];
})
```

The preceding code block represents the controller definition of `PlaylistsCtrl`. The first thing we need to do is to clear all the code within the controller. Basically, we need to delete all the code found within the controller. If you have done this correctly, your controller should now look similar to what we have in the following screenshot:

```
.controller('PlaylistsCtrl', function($scope) {
})
```

With that done, we can now begin to create the code to query our local JSON database with the angular `$http` service. The first thing we need to do to achieve this is to first add the dependency of our `$http` service to our controller. This step is very important as if we do not add this dependency correctly, our app will not load. To do this, simply add `$http` as the second parameter in the anonymous function part of your controller definition. If you have done this correctly, you should see something similar to what I have in the following screenshot:

```
44   .controller('PlaylistsCtrl', function($scope, $http) {
45
46 })
```

With that done, we can now go ahead and start writing the code to grab our data from our local database. To start this process, simply write the following code into your controller:

```
$scope.sites = [];
$http.get('/sites.json')
.then(function (response) {
  $scope.sites = response.data;
});
```

If you have done this correctly, your code should look very close to what we have in the following screenshot:

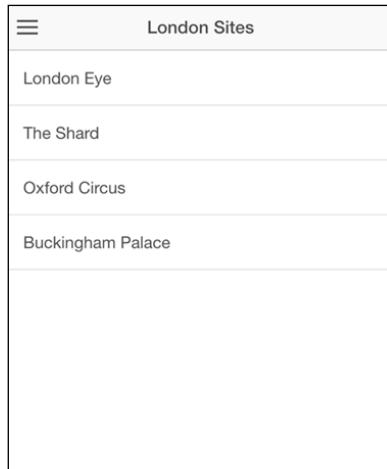
```
.controller('PlaylistsCtrl', function($scope, $http) {
  $scope.sites = [];
  $http.get('/sites.json')
    .then(function(response) {
      $scope.sites = response.data;
    });
})
```

At this point, I will explain what this block of code is doing. We start by simply initializing the variable `sites` as an array to the `$scope`. It is a good practice to always initialize your Angular `$scope` variables before using them. The next thing we try to do is make an Ajax get request using the shorthand `$http.get()` method. This `$http.get` method returns a promise so we handle this promise by using the `.then()` method of promise handling of Angular. In the promise handler function, you can see that we start by setting the `data` property of the `response` from the promise (`response.data`). This `data` property of the promise `response` (`response.data`) is the property that holds any data returned which in our case is the data from our `sites.json` file.

 One thing that might be a bit confusing is the fact that, for the first parameter of the `$http.get()` function, which takes the URL of the API or the file we want to consume, we have provided the following relative path '`/sites.json`'. You might be wondering why we have not correctly given a path relative to the `controller.js` file. This is because when working with Angular, all paths are referenced from the root `index.html` file. In our case, the `sites.json` and `index.html` files are in the same directory under the `www` directory, which is why we do not have the path '`../sites.json`', and instead have the path '`/sites.json`'.

With all this done, we have completed the process of creating our LTA application. All that is left now is to run the application. Go ahead and run this application using the `ionic serve` technique learned from *Chapter 1, First Look at Ionic*. Make sure you run this command from the root directory of your LTA app project.

If you have done this correctly, you should see a list of our tourist destinations as shown in the following screenshot:



## Summary

In this chapter, we learned how to create an Ionic application based on the side menu template. We used this knowledge to create our London Tourist Application. We also had a look at the code that makes up an Ionic side menu template and learned about the building block elements of a side menu application. We rounded up by querying some data via Ajax using the Angular `$http` service and rendered our tourist destinations.

In the next chapter, we will extend our existing application and use some more complex Ionic components to do some really cool stuff.

# 6

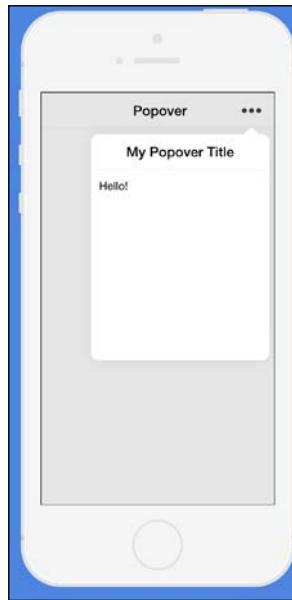
## Advanced Ionic Components

In this chapter, we will extend the application we created in *Chapter 5, The London Tourist App*. We are going to learn how to add some more complex features like the Ionic Popover and the Ionic Modal components to our current application. At the end of this chapter, we will have a popover menu and a modal window as part of our application.

### The Ionic Popover

The Ionic Popover component allows us to add a popover menu to our application. A popover menu is a contextual menu that is used to provide a hidden menu or extra menu options. It is normally used when we have limited space and want to present a list of options. Instead of cramming our limited available space, we create some sort of button so that, when clicked, the popover menu can pop up and show these menu items.

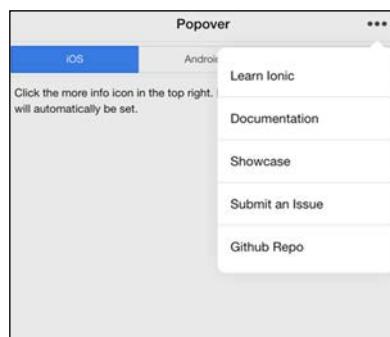
The following screenshot shows a good description of what a popover does in reality:



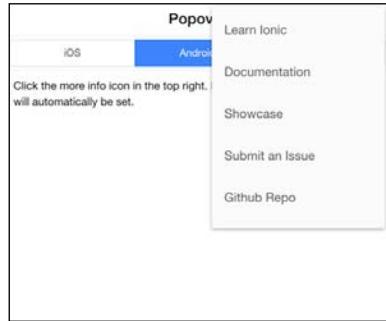
## Implementing the popover

We are going to implement our popover in our already existing application. The first thing you should do is open your application, as you have left the London Tourist Application in the previous chapter. What we will be aiming to do is create a popover that has three extra options as a list. These three options are **About**, **Help** and **Logout**. These three options will not perform any action as we will only be placing them for the sake of example. The following screenshots show a sample of what we will be aiming to achieve.

- For iOS:



- For Android:



To begin implementing our popover, open the `playlists.html` file of your LTA application project. Remember that this `playlists.html` file can be found by navigating to the `www` folder and looking into the `templates` folder within it. Here is the path: `www/templates/playlists.html`.

Now, you should have a file that closely resembles the following code:

```
<ion-view view-title="London Sites">
  <ion-content>
    <ion-list>
      <ion-item ng-repeat="site in sites">
        {{site.name}}
      </ion-item>
    </ion-list>
  </ion-content>
</ion-view>
```

## Adding the menu button

The first thing we are going to do is add the menu button that we want to trigger for our popover. This menu will display the popover when tapped. The following code block represents the code for button icon of our popover:

```
<ion-nav-buttons side="right">
  <button class="button button-clear icon ion-more"></button>
</ion-nav-buttons>
```

You are to replicate the preceding code just after the opening `<ion-view>` tag of your `playlists.html` page. The preceding code is using the `<ion-nav-buttons>` element to specify that we want to place a navigation button in our header. This element also has a `side` attribute with the value `right`. This `side` attribute is there to tell the `<ion-nav-buttons>` element which side of the page title to position itself. Within the `<ion-nav-buttons>` element is a simple button with some ionic styles that ensure that the button has an icon (`ion-more`) as our popover icon. If you have followed all the steps and replicated the code block correctly, your code should closely resemble the following code block:

```
<ion-view view-title="London Sites">

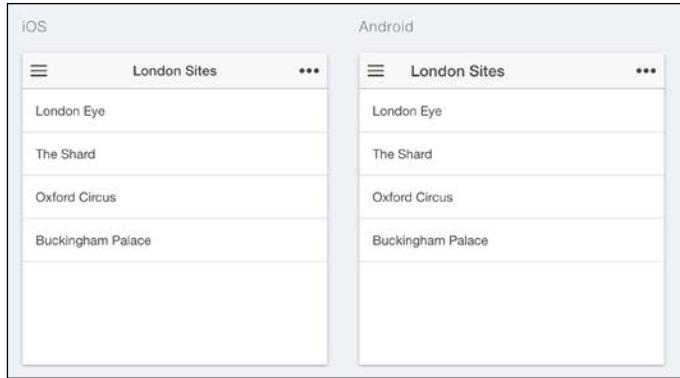
    <ion-nav-buttons side="right">
        <button class="button button-clear icon ion-more"></button>
    </ion-nav-buttons>

    <ion-content>
        <ion-list>
            <ion-item ng-repeat="site in sites">
                {{site.name}}
            </ion-item>
        </ion-list>
    </ion-content>
</ion-view>
```

At the moment, it is probably a good idea to test our application and see how our icon button looks. Fire up your application on a Chrome browser using the `ionic serve` technique as we have done in the past. Your screen should look close to what we have in the following screenshot.



If your title is centered to the left when using an Android emulator on Chrome, this is perfectly normal. The iOS equivalent will be centered.



## Coding the popover

The next step is to write the actual logic for our popover menu. The first thing we need to do is go into our `controller.js` file. This file can be found by looking in the following path from the root of your project:

```
www/js/controller.js
```

Within the `controller.js` file, locate the `PlaylistsCtrl` controller. It is within this controller that we will be implementing our popover, as it is the controller associated with our `playlists.html`.

## Adding the \$ionicPopover service

In order to use the Ionic Popover, Ionic has a special service called `$ionicPopover` that makes this very easy. Add `$ionicPopover` as a dependency by specifying it as a parameter on your `PlaylistsCtrl` controller. If you have done this correctly, your `PlaylistsCtrl` controller should now look similar to the following code:

```
.controller('PlaylistsCtrl', function($scope, $http,
$ionicPopover) {
    $scope.sites = [];
    $http.get('/sites.json')
        .then(function (response) {
            $scope.sites = response.data;
        });
})
```

## Finishing the popover code

The next step is to write the actual code to create the popover using the `$ionicPopover` service, as shown in the following code:

```
$ionicPopover.fromTemplateUrl('templates/popover.html', {
  scope: $scope
}).then(function(popover) {
  $scope.popover = popover;
});

$scope.openPopover = function($event) {
  $scope.popover.show($event);
};
```

The preceding code block uses the `$ionicPopover` service to instantiate a new popover. We also use the `.fromTemplateUrl` function of `$ionicPopover` to create the popover. This function allows us to pass a URL for a file that contains the HTML for our popover. The `.fromTemplateUrl` function also returns a promise which returns the instance of a popover created. We then bind this popover instance to our scope so that it is available for use in our view. There is, however, one small part that we have not done. We passed in a file path `templates/popover.html` as the file which contains the code for our popover. However, this `popover.html` file does not currently exist so we need to create it.

## Creating the popover.html file

To create our `popover.html` file, create a new file called `popover.html` under the `templates` folder. This `templates` folder can be found under the `www` folder located in the root directory of your project. Here is the path: `www/templates/popover.html`.

Now that we have created this file, the next step is to populate this file. Remember that what we are trying to achieve is to have a list of menu items in `popover.html`. We want these three options to be **About**, **Help**, and **Logout** to mimic a fake set of popover options.

To start creating the content of our popover, replicate the following code block into your `popover.html`:

```
<ion-popover-view>
  <ion-content>
    <div class="list">
      <b class="item" href="#">
        About
      </b>
      <b class="item" href="#">
```

```
        Help
    </b>
    <b class="item" href="#">
        Logout
    </b>
</div>
</ion-content>
</ion-popover-view>
```

If you have done this, you have completed implementing the template of your popover. Now, let's understand what the HTML code we just implemented on our `popover.html` file does. The `<ion-popover-view>` element is an element that is essential for indicating that this particular view is a popover. It also contains an `<ion-content>` element which is a container for all the visible parts of our view, or popover in this case. We then put a `div` tag with a class `list` which is one of the Ionic's built-in classes. Within this `div`, there are three HTML bold tags that represent our three fake options. That is all we need to complete the implementation for our template. The final step is to wire our popover to ensure it works as it should.

## Wiring up the popover

This is the final step to get our popover to work. Remember that we created a function on our `PlayListsCtrl` controller called `openPopover()` which takes in a `$event` parameter. This function will initiate the popover when executed. We will also have to pass the `$event` parameter, which is a reserved parameter that represents an event sent from the view.

To put this into action, we will first need to wire this `openPopover()` function to be executed when the popover icon we created earlier is clicked. This popover button is in our `playlists.html` file from earlier steps. Your current `playlists.html` file should look close to what we have in the following code block:

```
<ion-view view-title="London Sites">

    <ion-nav-buttons side="right">
        <button class="button button-clear icon ion-more"></button>
    </ion-nav-buttons>

    <ion-content>
        <ion-list>
            <ion-item ng-repeat="site in sites">
                {{site.name}}
            </ion-item>
        </ion-list>
    </ion-content>
</ion-view>
```

What we need to do is add an Ionic tap event on the popover icon button that we created. We can do this with the Ionic provided attribute directive called `on-tap`. This `on-tap` attribute directive takes in a function which we want to be executed when the containing element is tapped. In our case, we want this function to be the `openPopover` function. Right now our popover Icon button code looks as follows:

```
<ion-nav-buttons side="right">
  <button class="button button-clear icon ion-more"></button>
</ion-nav-buttons>
```

Now, the code for the `on-tap` ionic attribute directive for `<button>` will look as follows:

```
on-tap="openPopover($event)"
```

You can see `$event` being passed as a parameter. Remember that this is very important and must be passed exactly as that. The final code for your `playlists.html` will look like the following code block:

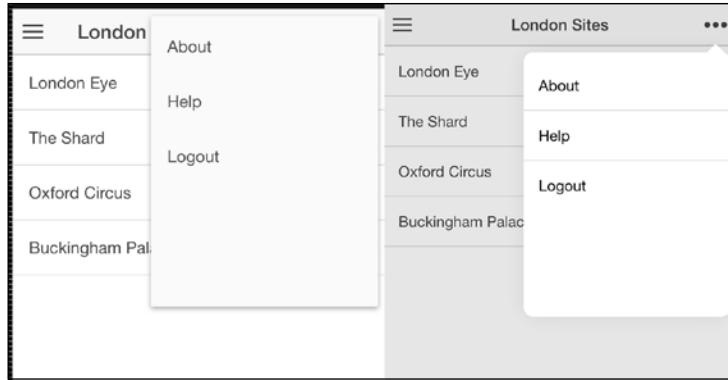
```
<ion-view view-title="London Sites">

  <ion-nav-buttons side="right">
    <button class="button button-clear icon ion-more" on-
      tap="openPopover($event)"></button>
  </ion-nav-buttons>

  <ion-content>
    <ion-list>
      <ion-item ng-repeat="site in sites">
        {{site.name}}
      </ion-item>
    </ion-list>
  </ion-content>
</ion-view>
```

With that done, we have completely finished the implementation of our popover. Now, we can run it in our browser using the `ionic serve` technique to see what it looks like.

If you correctly ran your app using the `ionic serve` technique, you should see something that looks like the following screenshot when you click the popover icon button. The view will be different depending on whether you are testing with an Android or iOS emulator setting:



The popover is a great contextual menu tool for fitting extra menu options. It also has an automatic way of displaying a different version depending on what mobile operating system it is being displayed on. Now that we have learned how to use the Ionic Popover, let's learn to use the Ionic Modal.

## The Ionic Modal

The Ionic Modal is a component feature that Ionic provides and is used to create a modal window in our application. A modal window is a view that pops up on an existing page without losing the context of your current action. As soon as it is dismissed, the previous view state is restored. It is a great tool for collecting extra information or displaying something on the screen without losing track of our current state.

## Creating the modal

Ionic exposes the modal functionality via a service called the `$ionicModal` service. This service provides us a way of creating a modal in our application. Before we begin implementing our modal, let's understand what we aim to do with the modal feature in our application.

We will still be using our LTA application and adding a modal. We want this modal to mimic a sample **About** page of our application which will have some small details about the app. Remember that we already have a button from the popover we created earlier which has a text labeled as **About**. We will wire this popover item to simply open the modal when tapped.

## Implementing the modal

To begin implementing the modal, open your `controller.js` file and locate the `PlaylistsCtrl` controller. The first thing to do is add a dependency to the `$ionicModal` service on the `PlaylistsCtrl` controller. This is done by adding `$ionicModal` as a parameter for the `PlaylistsCtrl` controller function definition. Doing this correctly should make your `PlaylistsCtrl` controller look like what we have in the following code block:

```
.controller('PlaylistsCtrl', function($scope, $http,
  $ionicPopover, $ionicModal) {
  $scope.sites = [];
  $http.get('/sites.json')
    .then(function (response) {
      $scope.sites = response.data;
    });

  $ionicPopover.fromTemplateUrl('templates/popover.html', {
    scope: $scope
  }).then(function(popover) {
    $scope.popover = popover;
  });

  $scope.openPopover = function($event) {
    $scope.popover.show($event);
  };
})
```

The next thing we are going to do is write the code for our modal in our `PlaylistsCtrl` controller. The following code represents the code for our modal:

```
$ionicModal.fromTemplateUrl('templates/modal.html', {
  scope: $scope
}).then(function(modal) {
  $scope.modal = modal;
});
$scope.openModal = function() {
  $scope.modal.show();
};
$scope.closeModal = function() {
  $scope.modal.hide();
};
```

Replicate the preceding code into your `PlaylistsCtrl` controller. If you have done this correctly, your code block for the `PlaylistsCtrl` controller should look like the following:

```
.controller('PlaylistsCtrl', function($scope, $http,
  $ionicPopover, $ionicModal) {
  $ionicModal.fromTemplateUrl('templates/modal.html', {
    scope: $scope
  }).then(function(modal) {
    $scope.modal = modal;
  });

  $scope.openModal = function() {
    $scope.modal.show();
  };

  $scope.closeModal = function() {
    $scope.modal.hide();
  };

  $scope.sites = [];
  $http.get('/sites.json')
    .then(function (response) {
      $scope.sites = response.data;
    });

  $ionicPopover.fromTemplateUrl('templates/popover.html', {
    scope: $scope
  }).then(function(popover) {
    $scope.popover = popover;
  });

  $scope.openPopover = function($event) {
    $scope.popover.show($event);
  };
})
```

Now, let's understand what the code for the modal is doing. We used the `$ionicModal` service to create a modal via its `.fromTemplateUrl()` method. This method takes two parameters; the first being the path to an HTML file containing the modal, and the second being an `options` object. This `options` object lets us customize the modal and even provides us with ways to customize things like what animation to use. For now, we only specify the scope the modal should use, which in this case is the scope of our controller.

The `.fromTemplateUrl` method returns a promise with the created modal, which we set to our `$scope`. The following code is a reflection of the modal creation:

```
$ionicModal.fromTemplateUrl('templates/modal.html', {
  scope: $scope
}).then(function(modal) {
  $scope.modal = modal;
});
```

We also have two functions that we created. These functions are `.openModal()` and `.closeModal()`. The `openModal()` function is bound to the `$scope`, and all it does is use the created modal's `.show()` method. The `.closeModal()` function does the opposite by implementing the `.hide()` method of the created modal. One thing we have not done yet is create the HTML template we passed, which is the `modal.html` in this case.

## Creating the modal.html file

Navigate to your `templates` folder and create a new HTML file called `modal.html`. The following code represents the template file for our modal, and you are to replicate this code into your `modal.html` file:

```
<ion-modal-view>
  <ion-header-bar class="bar bar-header bar-positive">
    <h1 class="title">About The App</h1>
    <button class="button button-clear button-primary" on-
      tap="closeModal()">Cancel</button>
  </ion-header-bar>

  <ion-content class="padding">
    The LTA app is part of the Ionic By Example book written
    by Sani Yusuf.
  </ion-content>
</ion-modal-view>
```

If you look at this code closely, you can see an `<ion-modal-view>` element as the root element of the code. This `<ion-modal-view>` element is the root element of any modal template. We can also see that we have an `<ion-header-bar>` element and this element has a `<h1>` element used to declare the title of the modal header. There is also a `<button>` element that has an `on-tap` attribute that is directed to a `closeModal()` function which we created earlier.

There is also an `<ion-content>` element which is used to contain the visible main body of the modal. There is some dummy text to mimic the **About** page of the LTA app, but feel free to add some of your own HTML text. The last step we need to do is wire our popover button to open our modal.

## Wiring up the modal

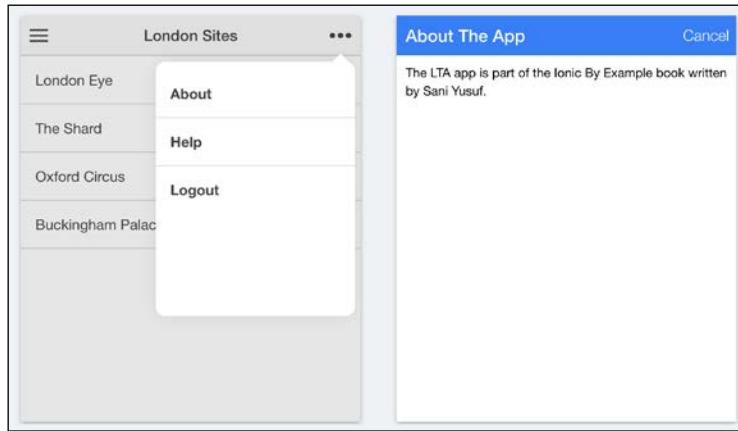
To wire up our modal, remember that we want our **About** popover menu item to open the modal when tapped. To begin, first open the `popover.html` file of your project. What you have currently is as follows:

```
<ion-popover-view>
  <ion-content>
    <div class="list">
      <b class="item">
        About
      </b>
      <b class="item">
        Help
      </b>
      <b class="item">
        Logout
      </b>
    </div>
  </ion-content>
</ion-popover-view>
```

All we need to do is use the Ionic `on-tap` attribute on the `About` entry to reference the `openModal()` function. Doing this correctly will make our popover code look like the following:

```
<ion-popover-view>
  <ion-content>
    <div class="list">
      <b class="item" on-tap="openModal()">
        About
      </b>
      <b class="item">
        Help
      </b>
      <b class="item">
        Logout
      </b>
    </div>
  </ion-content>
</ion-popover-view>
```

With this done, we have completed the implementation of our modal window. The next thing to do is to go ahead and test this. To do this, run your application using the `ionic serve` technique. When your app is up and running in the browser, tap the popover icon and the **About** option. This should bring up a modal window like the one shown in the following screenshot:



## Summary

In this chapter, we used two very important features of Ionic and learned to create a popover and modal. We still used our LTA application from the previous chapter. The Ionic Popover is a great feature which is used to add extra menu items or provide contextual menu options. We also learned about the Ionic Modal, which is used to provide a view over another view of the app while maintaining the context.

In the next chapter, we will learn to use some of the customization techniques of Ionic, along with how to customize our Ionic app.

# 7

## Customizing the App

In the previous chapter, we dug deeply into some more advanced features of Ionic like the popover and the modal features. In this chapter, we will be focusing on customizing an Ionic application. The Ionic SDK comes by default with some great tools that make it easy to customize your application to fit the design guides of your brand. This is thanks to its built-in integration of Gulp for your build process needs and SCSS for CSS preprocessing.

Ionic also has a special Angular provider called `$ionicConfigProvider`. This provider can be used to do a lot of configuration and customization like specifying what type of animations your application should use or even more advanced stuff like specifying how many cache items you want in your cache. The `$ionicConfigProvider` also lets you specify these configurations on a global level, or on a platform-by-platform basis.

### Customizing the look and feel of your app

When you created an Ionic application using one of the Ionic templates, you would have noticed by now that it comes with some built-in default CSS styles. Many times you will want to know how you can add your own colors and styles while keeping some of the built-in Ionic styles.

## Ionic styles with SASS

This is well thought out by the Ionic team and for this reason, they actually created all their CSS styles using SCSS. SCSS is an independent technology based on SASS that lets you write CSS in an object-oriented way which then gets compiled into CSS. SCSS is a really cool way to write CSS rules as it allows us to create variables and use them to create our style sheet. If you are completely new to SCSS and you want to see some brief information about SCSS, feel free to visit <http://sass-lang.com>.

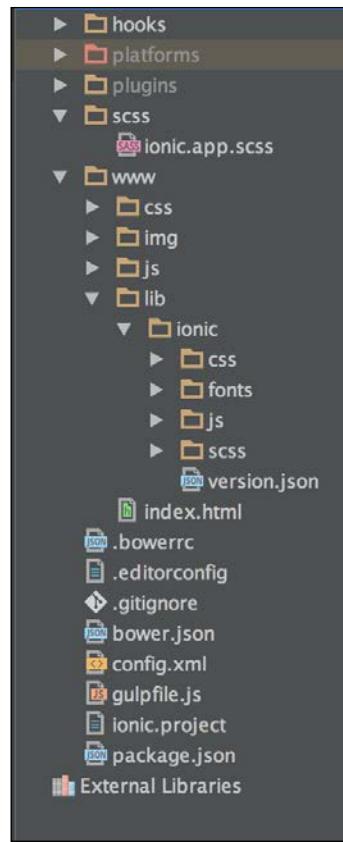
Now, let's have a look at the folder structure of an Ionic project once more with customization of our styles with SCSS in mind.

## Ionic SCSS overview

To have an overview of the SCSS structure of Ionic, we are going to create a brand new application using the Ionic blank template. We are going to call this application `custom-app`. The following is the command to create this new application. Fire up a terminal window on your computer and CD into a directory of your choice and run the following command:

```
ionic start custom-app blank
```

After you have created your new `custom-app` application, open this new project in your favorite IDE to have an overview of the folder structure. You should see something close to what we have in the following screenshot:



There are two folders that you should pay close attention to. The first folder is the `scss` folder found in the root directory of the project. This folder has a file called `ionic.app.scss` within it; we will take a look at this in more detail. The following is a screenshot of what this folder looks like:

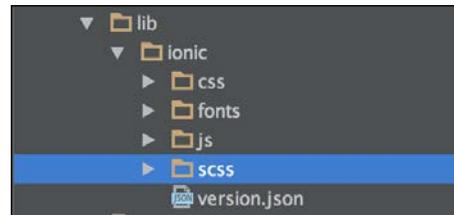


The second folder is also titled `scss`, but this folder can be found by navigating to the following path from the root folder `www/lib/ionic/scss`.

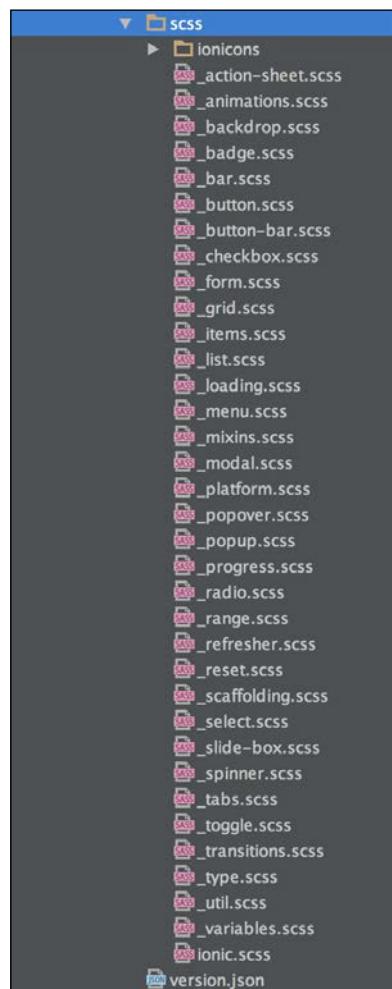
## *Customizing the App*

---

The following screenshot shows this folder:



Now, if you look even further within this second `scss` folder, you should see something that closely resembles what we have in the following screenshot with a number of SCSS files within the `scss` folder:



This `scss` folder contains a numerous amount of files and you might be wondering what these files are for. As a part of this book, you do not need to understand the entire process of what these files are doing, but you need to know that they are the files that contain the SCSS code for every Ionic element. The entire Ionic CSS style sheet is generated by compiling these SCSS files. It is possible to go into these files to make changes to any SCSS file, but this is probably not a good idea, as you will risk breaking any dependencies in the SCSS code. For this reason, Ionic provides a much simpler way to do this thanks to `ionic.app.scss` that we briefly looked at earlier and will be looking at closely now.

## The `ionic.app.scss` file

The `ionic.app.scss` file can be found within a directory called `scss` in the project root directory, as shown in the following screenshot:



This file is the most important file for customizing the styles of your Ionic app. Think of this file as an interface for overriding any SCSS style contained in any of the SCSS files we noted in the `www/lib/ionic/scss` path. If you look at this `ionic.app.scss` file currently, it should look like what we have in the following code:

```
/*
To customize the look and feel of Ionic, you can override the
variables in ionic's _variables.scss file.
```

For example, you might change some of the default colors:

```
$light: #fff !default;
$stable: #f8f8f8 !default;
$positive: #387ef5 !default;
$calm: #11c1f3 !default;
$balanced: #33cd5f !default;
$energized: #ffc900 !default;
$assertive: #ef473a !default;
$royal: #886aea !default;
$dark: #444 !default;
*/
```

```
// The path for our ionicon's font files, relative to the built CSS in
www/css
```

```
$ionicons-font-path: "../lib/ionic/fonts" !default;  
  
// Include all of Ionic  
@import "www/lib/ionic/scss/ionic";
```

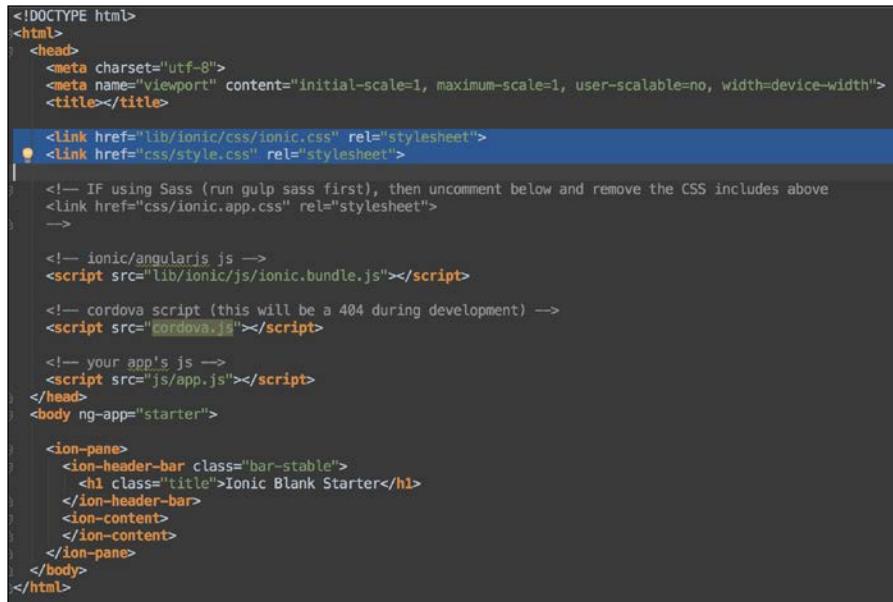
From the preceding code, you can even see some comments that tell you how to use the file to override your SCSS styles. Now, before we start learning how to actually override these files, first let's learn how to set up our SCSS for our project.

## Setting up SCSS

Before we set up the SCSS, we will first have a brief look at how our CSS is currently integrated. When you create a new Ionic project, the project uses styles from two sources by default.

The first source is the `ionic.css` file which can be found in the path `lib/ionic/css/ionic.css`. This file contains already compiled CSS code for all the Ionic default styles. It is simply a CSS compilation of all the SCSS files found in the `www/lib/ionic/scss/ionic` directory relative to the root directory of your project.

The second source is the `style.css` file found in the `css/style.css` path relative to the root directory of your project. This file is normally empty at the time you create your project and is a place where you can enter your own custom styles in CSS, if you do not want to use SCSS. A look at the `index.html` file as shown in the following screenshot shows how these two files are referenced as CSS style sheets by default:



A screenshot of a code editor showing the `index.html` file of an Ionic project. The file contains the following HTML structure:

```
<!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">

    <!-- If using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">

    <ion-pane>
      <ion-header-bar class="bar-stable">
        <h1 class="title">Ionic Blank Starter</h1>
      </ion-header-bar>
      <ion-content>
      </ion-content>
    </ion-pane>
  </body>
</html>
```

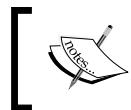
The line `<link href="css/style.css" rel="stylesheet">` is highlighted with a blue background, indicating it is the file being discussed in the text.

With this explained, we will go ahead and start setting up SCSS on our Ionic application.

Setting up SCSS can be quite challenging traditionally but Ionic comes built-in with some tools that make it easy. To begin the process of setting up the SCSS of your project, fire up a terminal window and simply navigate into your project's root directory by running the `cd custom-app` command.

The next step is to install bower on your computer if you do not already have this installed. You can do so by running the following command:

```
npm install bower -g
```



You might need to prefix the `sudo` command if you are on a Linux or Mac computer. This will be `sudo npm install bower -g`.



After this, the final step to get SCSS setup is by running the following command:

```
ionic setup sass
```

This command will do all the necessary things behind the scenes that are needed to enable your project to work with SCSS. After this command is complete, you will notice a new folder called the `node_modules` folder in the root of your project. This is completely normal and is the folder that contains the packages necessary for your project to work with SCSS.

By now, we have successfully set up SCSS for our project. The first thing you should look at is your `index.html` file. Your `index.html` should resemble the following code block:

```
<!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>

    <!-- compiled css output -->
    <link href="css/ionic.app.css" rel="stylesheet">

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>
```

## *Customizing the App*

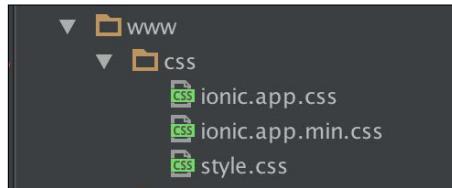
---

```
<!-- cordova script (this will be a 404 during development)
-->
<script src="cordova.js"></script>

<!-- your app's js -->
<script src="js/app.js"></script>
</head>
<body ng-app="starter">

<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Ionic Blank Starter</h1>
  </ion-header-bar>
  <ion-content>
  </ion-content>
</ion-pane>
</body>
</html>
```

The first thing you will notice in the header is that the reference to CSS files have changed in comparison to what we briefly discussed earlier. Now, you have only one CSS reference in the `<head>` part of `index.html` pointing to `css/ionic.app.css`. You might be wondering how this happened. Well, basically when you set up SCSS like we have done in this chapter so far, Ionic automatically sets up the SCSS to compile all the SCSS and output them into `ionic.app.css`.



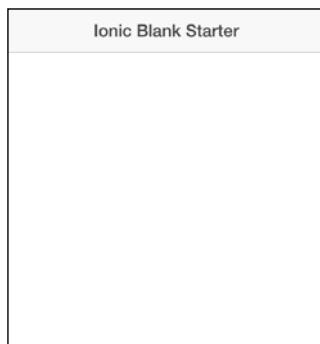
If you navigate to the `www/css` path, you will see that we have three files as opposed to one as we saw earlier. You will see an `ionic.app.css` file and an `ionic.app.min.css` file. These two files are the same with `ionic.app.min.css` being a minified version of the `ionic.app.css`. They are the output of all the SCSS files that we checked out earlier compiled into one file. There are a lot more things that happen behind the scenes to ensure that this SCSS compilation happens, but for the sake of simplicity we won't be going deep into that in this book.

## Customizing the SCSS

To begin customizing our app, the first thing you want to do is to run your application using the `ionic serve` technique learned from previous chapters in this book, using the following command:

```
ionic serve
```

This should bring up your application running in the browser and you should see something that closely resembles what we have in the following screenshot:



Make sure you don't close your terminal or terminate the serve session from here on, in order to follow the instructions that come soon.



Now to explain what we will try to do, first let's have a look at the code for the head of this app. The code block is the code for our app and you can find this in the `index.html` file in the `www` folder of your project:

```
<!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>

    <!-- compiled css output -->
    <link href="css/ionic.app.css" rel="stylesheet">

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>
```

## *Customizing the App*

---

```
<!-- cordova script (this will be a 404 during development)
-->
<script src="cordova.js"></script>

<!-- your app's js -->
<script src="js/app.js"></script>
</head>
<body ng-app="starter">

<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Ionic Blank Starter</h1>
  </ion-header-bar>
  <ion-content>
  </ion-content>
</ion-pane>
</body>
</html>
```

Pay close attention to the piece of code that represents the main view part of the preceding code block which is also represented in the following code block:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Ionic Blank Starter</h1>
  </ion-header-bar>
  <ion-content>
  </ion-content>
</ion-pane>
```

If you look at the opening `<ion-header>` tag, you will see that it has a class called `bar-stable`. This is an in-built class that Ionic comes with which gives the header a sort of light gray color, as seen from the screenshot we visited earlier.

Let's say we want to customize this header to fit our brand color and let's say, for example, that our brand color and this brand happens to be my favorite accent of red which has the hex code of `#D71300`.

Now, you might be tempted to go into the `ionic.app.css` file to look for every occurrence of this in our CSS style sheet and change it. But remember that this `ionic.app.css` is generated based on our SCSS files. Ionic gives us a great way to override default styles with SCSS thanks to the `ionic.app.scss` file which can be found in the `scss` directory. We looked at this file earlier and we are going to look at it again:

```
/*
To customize the look and feel of Ionic, you can override the
variables in ionic's _variables.scss file.
```

For example, you might change some of the default colors:

```
$light: #fff !default;
$stable: #f8f8f8 !default;
$positive: #387ef5 !default;
$calm: #11c1f3 !default;
$balanced: #33cd5f !default;
$energized: #ffc900 !default;
$assertive: #ef473a !default;
$royal: #886aea !default;
$dark: #444 !default;
*/  
  
// The path for our ionicons font files, relative to the built CSS  
in www/css  
$ionicons-font-path: "../lib/ionic/fonts" !default;  
  
// Include all of Ionic  
@import "www/lib/ionic/scss/ionic";
```

The preceding code block resembles what you currently have in your `ionic.app.scss` file. To override the color of the header, we will override the current color of the `$stable` variable of our SCSS.

The code for this is as follows:

```
$stable: #D71300;
```

You are supposed to replicate the preceding code anywhere but just before the last line of the following code block:

```
@import "www/lib/ionic/scss/ionic";
```

Now, your final code should resemble the following:

```
/*
To customize the look and feel of Ionic, you can override the
variables in ionic's _variables.scss file.
```

For example, you might change some of the default colors:

```
$light: #fff !default;
$stable: #f8f8f8 !default;
$positive: #387ef5 !default;
$calm: #11c1f3 !default;
$balanced: #33cd5f !default;
$energized: #ffc900 !default;
```

---

## *Customizing the App*

---

```
$assertive: #ef473a !default;  
$royal: #886aea !default;  
$dark: #444 !default;  
*/  
  
$stable: #D71300;  
  
// The path for our ionicons font files, relative to the built CSS  
in www/css  
$ionicons-font-path: "../lib/ionic/fonts" !default;  
  
// Include all of Ionic  
@import "www/lib/ionic/scss/ionic";
```

Once this is done, save the `ionic.app.scss` file. By doing this, you have completed the process of overriding the app, and your header should now be red. Go back to your application on the browser or run your app with the `ionic serve` technique if you don't have it running and you should see something that looks similar to what we have in the following screenshot:



You can see that header now takes the color of the hex code we provided in the `ionic.app.scss` file. We can override any default file with this file. All you need to do is have a glance through the `lib/ionic/scss` folder, identify the SCSS rule you want to override, and override in `ionic.app.scss`.

With this done, we have completed the process of learning how to override and set up SCSS of our Ionic app. The next step is to learn about `$ionicConfigProvider`.

## \$ionicConfigProvider

`$ionicConfigProvider` is a provider that Ionic exposes and which allows us to do some very powerful configurations. We will not be writing any code for this as it is an advanced feature but you should be well aware of its existence.

Some of the features that `$ionicConfigProvider` lets you do, include the following:

- Specify the transition type for your app
- Set the maximum cache
- Disable/enable animations
- Enable/enable native scrolling
- Specify tabs positions

These and many more are some of the features that `$ionicConfigProvider` lets you fiddle with. Remember that this feature is a fairly advanced feature and it is very likely possible to completely design your app without it. Most apps most likely do not use its features but if you find yourself ever needing to use it, you can visit the official documentation for `$ionicConfigProvider` to see its full potential at [http://ionicframework.com/docs/api/provider/\\$ionicConfigProvider/](http://ionicframework.com/docs/api/provider/$ionicConfigProvider/).

## Summary

In this chapter, we learned how to customize our application by setting up SCSS for our Ionic app. We also had a brief look at `$ionicConfigProvider` and saw some of its wonderful features. In the next chapter, we will get to learn how to create a new type of Ionic app based on the tabs template.



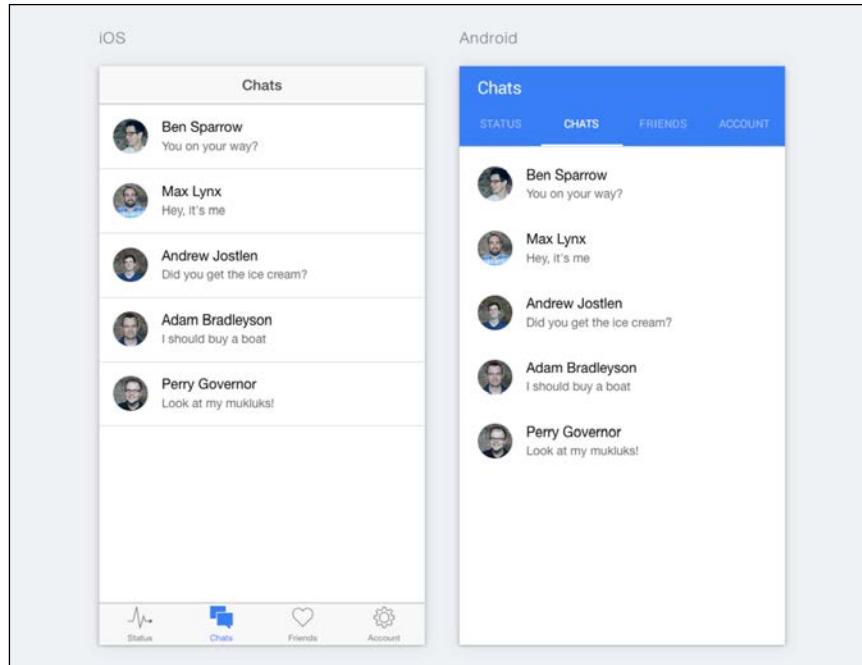
# 8

## Building a Simple Social App

In this chapter, we are going to focus on learning how to create an Ionic application that has tabs using the Ionic tabs template. We will also have a look at some of the things that make up the tabs template and learn how to add features into it.

### The Ionic tabs application

Tabs are a very common menu system in mobile apps. They provide users with a simple yet effective way to create independent views in an app that sort of act like apps within an app.



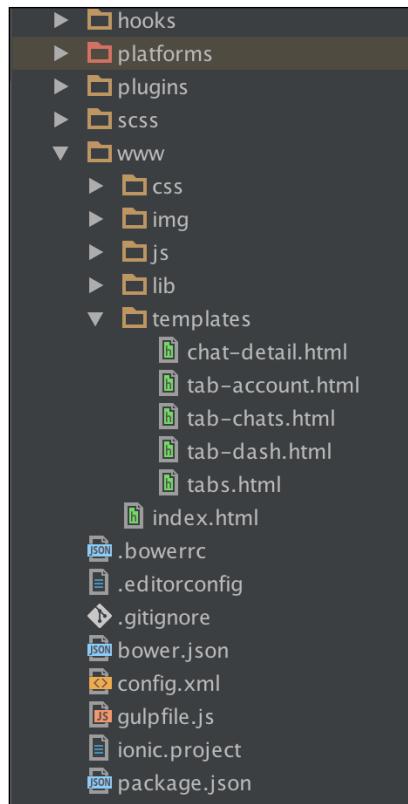
The preceding screenshot is a view of a sample Ionic tabs application. One great feature that a tabbed menu system provides is the ability to maintain the independent context within each individual tab menu. No matter where you are in the application, you always have the option of switching to another tab at any point. Navigation history is another feature that the tabs menu provides. You are able to navigate to different views within each tab, and you do not lose this navigation history when you switch back and forth between any tab menu. Now that we have some clarity about what the tabs application entails, let's go ahead and create a brand new tab application and look in detail at how it operates.

## **Creating an Ionic tabs application**

Creating an Ionic tabs application is not too different from creating the side menu and blank Ionic applications as we have done in the previous chapters of this book. We are going to create a new Ionic tabs application, and we will call this application `tabs-app`. To create this new application, fire up a terminal window and run the following command:

```
ionic start tabs-app tabs
```

Using the preceding command, you will create your `tabs-app` ionic application successfully. The next thing we are going to do is to have an overview of the application we just created. To do this, simply open the `tabs-app` project in your favorite IDE. You should have a projects folder structure that looks similar to what I have in the following screenshot:



## Running the tabs-app application

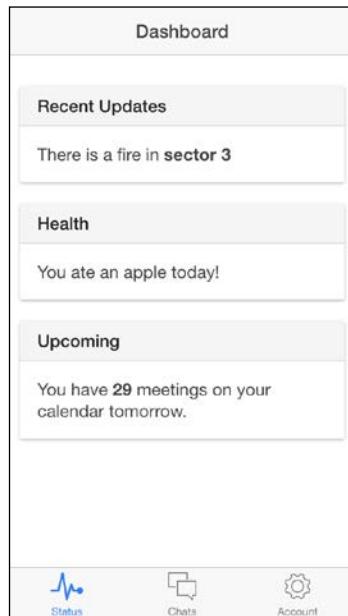
Now that we have created our app, let's go ahead and see it in action. To do this, fire up a terminal window on your computer and run your application using the `ionic serve` technique.



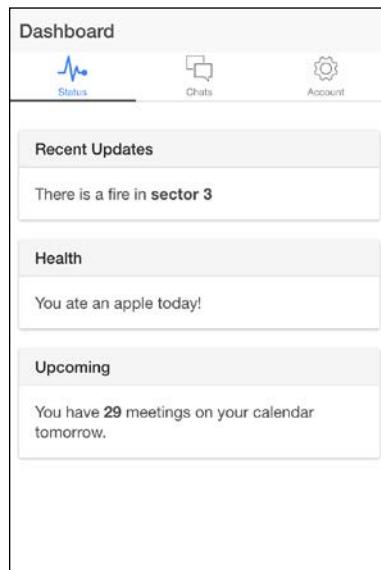
Make sure you are within your project's folder by running `cd tabs-app`. To run your app using the `ionic serve` technique, run the `ionic serve` command on your terminal.

You should see an application with three tabs that looks similar to what we have in the following screenshots.

- For iOS:



- For Android:



## Overview of the tabs-app application

To begin to understand the life cycle of our `tabs-app` Ionic tabs application, we first need to have a look at the entry module of our application. Our entry module is normally specified within the `index.html` file of our app via the `ng-app` directive.



The `index.html` file is located in the `www` directory of your application.



A look through your `index.html` will reveal a file that closely resembles what we have in the following screenshot:

```
<!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">

    <!-- If using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/services.js"></script>
  </head>
  <body ng-app="starter">
    <!--
      The nav bar that will be updated as we navigate between views.
    -->
    <ion-nav-bar class="bar-stable">
      <ion-nav-back-button>
        </ion-nav-back-button>
    </ion-nav-bar>
    <!--
      The views will be rendered in the <ion-nav-view> directive below
      Templates are in the /templates folder (but you could also
      have templates inline in this html file if you'd like).
    -->
    <ion-nav-view></ion-nav-view>
  </body>
</html>
```

You will see an Angular module called `starter` specified as on the opening `<body>` tag of our page via the `ng-app` directive. This can be seen highlighted in the preceding screenshot. This `starter` module is normally located in our `app.js` file, and we are going to have a look at it to understand the module even more deeply.



The `app.js` file is located in the `www/js` path of your project.



Open your `app.js` file and pay close attention to the `.config()` function where your routes are configured. Pay close attention to the first route definition of a route called `tab`. This route definition is represented in the following screenshot:

```
① .config(function($stateProvider, $urlRouterProvider) {  
②   // Ionic uses AngularUI Router which uses the concept of states  
③   // Learn more here: https://github.com/angular-ui/ui-router  
④   // Set up the various states which the app can be in.  
⑤   // Each state's controller can be found in controllers.js  
⑥   $stateProvider  
⑦     // setup an abstract state for the tabs directive  
⑧     .state('tab', {  
⑨       url: '/tab',  
⑩       abstract: true,  
⑪       templateUrl: 'templates/tabs.html'  
⑫     })
```

This `tab` state is an abstract state. An **abstract state** in Angular is a state that you cannot directly navigate to but which can contain child states that can be navigated to. This is a great way to create some sort of hierarchy for your states.

Based on the state definition of the tabs as highlighted in the preceding screenshot, you can see that it references `templateUrl` to the `tabs.html` file contained in the `templates/template.html` directory. To understand how Ionic works with tabs, let's explore the `tabs.html` file.

## Overview of the `tabs.html` file

When you open your `tabs.html` file, you will see something that closely resembles what I have in the following screenshot:

```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">  
  <!-- Dashboard Tab -->  
  <ion-tab title="Status" icon-off="ion-ios-pulse" icon-on="ion-ios-pulse-strong" href="#/tab/dash">  
    <ion-nav-view name="tab-dash"></ion-nav-view>  
  </ion-tab>  
  
  <!-- Chats Tab -->  
  <ion-tab title="Chats" icon-off="ion-ios-chatboxes-outline" icon-on="ion-ios-chatboxes" href="#/tab/chats">  
    <ion-nav-view name="tab-chats"></ion-nav-view>  
  </ion-tab>  
  
  <!-- Account Tab -->  
  <ion-tab title="Account" icon-off="ion-ios-gear-outline" icon-on="ion-ios-gear" href="#/tab/account">  
    <ion-nav-view name="tab-account"></ion-nav-view>  
  </ion-tab>  
  
</ion-tabs>
```

You will clearly see that the entire markup is wrapped within the `<ion-tabs>` element. This `<ion-tabs>` element is the root element that acts like a container for the tabs that you declare in your Ionic tabs application. You can see that the opening `<ion-tabs>` tag also has a `class` attribute with some built-in Ionic CSS classes provided. This is because the `<ion-tabs>` element is just like every other element and is submissive to some CSS styling.

## The `<ion-tab>` element

Within the `<ion-tabs>` element, you will see three distinct `<ion-tab>` elements. The `<ion-tab>` element is the element used to create a tab and must be a child element of the `<ion-tabs>` element. You will see that each `<ion-tab>` element has some attributes. The `title` attribute is used to specify the title that that particular tab will display. The `icon-on` and `icon-off` are attributes that are used to define what icons get displayed when the tab is in focus and out of focus. Lastly, the `href` attribute is used to provide the path of the route that should be navigated to when that particular tab is selected.



There are a lot more attributes that are available for different customizations and actions for `<ion-tab>`, and these are all available and duly documented on the official Ionic documentation page.

Within each `<ion-tab>` element, you will find an `<ion-nav-view>` declaration. The `<ion-nav-view>` is an element used to refer to an Angular view. If you pay close attention, you will see that the `<ion-nav-view>` elements have a `name` attribute, which has values. This `name` attribute is used to specify the name of a particular view that is defined in our `app.js` file. If you have another short look at the `app.js` file, as we did previously in this chapter, you will see that some of the states have views defined. A clear demonstration of this is shown in the following screenshot of the `tab.dash` state:



```
.state('tab.dash', {
  url: '/dash',
  views: {
    'tab-dash': {
      templateUrl: 'templates/tab-dash.html',
      controller: 'DashCtrl'
    }
  }
})
```

You can see that there is a `tab-dash` view named within the `views` object, and this `tab-dash` view has a `templateUrl` definition as well as a `controller` definition similar to a normal state definition. This is how Ionic provides a hierarchy that enables each tab to have a separate `<ion-nav-view>`, where its view is placed. To get an even better understanding of how this tab system works, we will be adding another tab to our application.

## Adding tabs to the tabs-app application

We will add one new tab which will contain a feature that will let users post messages like a message board and see that it appears similar to a Facebook wall or a Twitter wall. We will be calling this new tab the `wall` tab. To add this new tab, the first thing we need to do is to add the route for our new tab.

### Adding the state for the new tab

To add the state for our new tab, we need to define this tab in our `app.js` file where all our default tab routes are defined. Within the `.config()` function found in your `app.js` file, place the following block of code just after the state definition of the `tab` abstract state:

```
.state('tab.wall', {
  url: '/wall',
  views: {
    'tab-wall': {
      templateUrl: 'templates/tab-wall.html',
      controller: 'WallController'
    }
  }
})
```

If you have done this correctly, parts of the `.config()` function of your `app.js` file should look something like this:

```
.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    // setup an abstract state for the tabs directive
    .state('tab', {
      url: '/tab',
      abstract: true,
      templateUrl: 'templates/tabs.html'
    })

    .state('tab.wall', {
      url: '/wall',
      views: {
        'tab-wall': {
          templateUrl: 'templates/tab-wall.html',
          controller: 'WallController'
        }
      }
    })
    // Each tab has its own nav history stack:

    .state('tab.dash', {
      url: '/dash',
      views: {
        'tab-dash': {
          templateUrl: 'templates/tab-dash.html',
          controller: 'DashCtrl'
        }
      }
    })
})
```

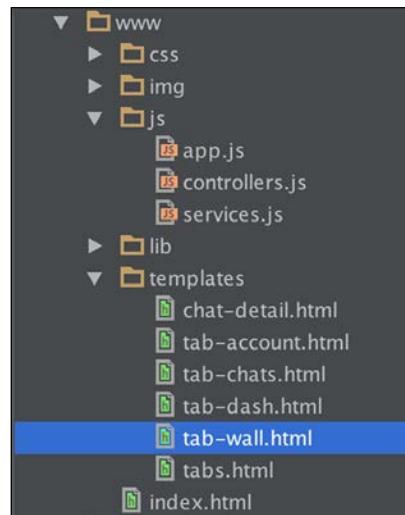
Let's try to understand what we have just done here. We have created a new state called `tab.wall`, which has a route `/tab`. This means that we are able to navigate to this `tab.wall` state or `/tab` route as part of our Angular application. We have also created a new view called `tab-wall`, and later in this chapter, we will use this `tab-wall` view to reference it as where we want the content of our newly created tab to be displayed.

If you take a closer look at our new state definition, you will see that we referenced a `templateUrl` to a file with the path `templates/tab-wall.html` and a controller, `WallController`, both of which we have not yet created. We will need to create this `tab-wall.html` file and also create the `WallController` controller.

## Creating the `tab-wall.html` file

To create the `tab-wall.html` file correctly, we need to make sure that we create it within the `templates` directory in order for it to match the `templates/tab-wall.html` directory which we passed when declaring our state definition.

Create a file called `tab-wall.html` within your `templates` folder. If you have done this correctly, your `templates` directory should look something very similar to what we have in the following screenshot:



The next step is to actually populate the newly created `tab-wall.html` file. Place the code as shown in the following code block into your `tab-wall.html` file:

```
<ion-view view-title="Wall">
  <ion-content class="padding">

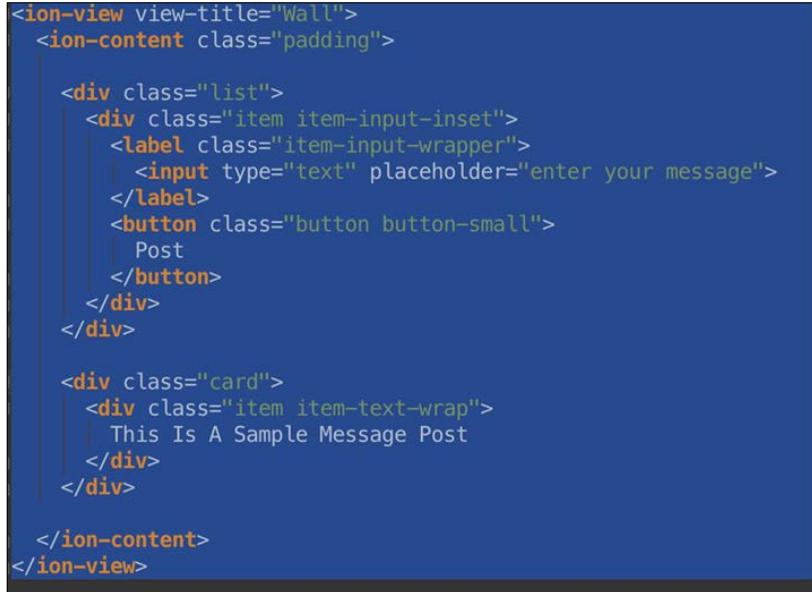
    <div class="list">
      <div class="item item-input-inset">
```

```
<label class="item-input-wrapper">
  <input type="text" placeholder="enter your message">
</label>
<button class="button button-small">
  Post
</button>
</div>
</div>

<div class="card">
  <div class="item item-text-wrap">
    This Is A Sample Message Post
  </div>
</div>

</ion-content>
</ion-view>
```

If you have correctly done this, your `tab-wall.html` should look something like the following screenshot:



This next step is to create the controller we defined in our state definition.

## Creating the WallController controller

To create the WallController controller, first we need to open the `controller.js` file. This file can be found within the same folder as our `app.js` file, that is, the `JS` folder. Your `controller.js` file should closely resemble what we have in the following screenshot:

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

.controller('DashCtrl', function($scope) {})

.controller('ChatsCtrl', function($scope, Chats) {
  // With the new view caching in Ionic, Controllers are only called
  // when they are recreated or on app start, instead of every page change.
  // To listen for when this page is active (for example, to refresh data),
  // listen for the $ionicView.enter event:
  //
  //$scope.$on('$ionicView.enter', function(e) {
  //});

  $scope.chats = Chats.all();
  $scope.remove = function(chat) {
    Chats.remove(chat);
  };
})

.controller('ChatDetailCtrl', function($scope, $stateParams, Chats) {
  $scope.chat = Chats.get($stateParams.chatId);
})

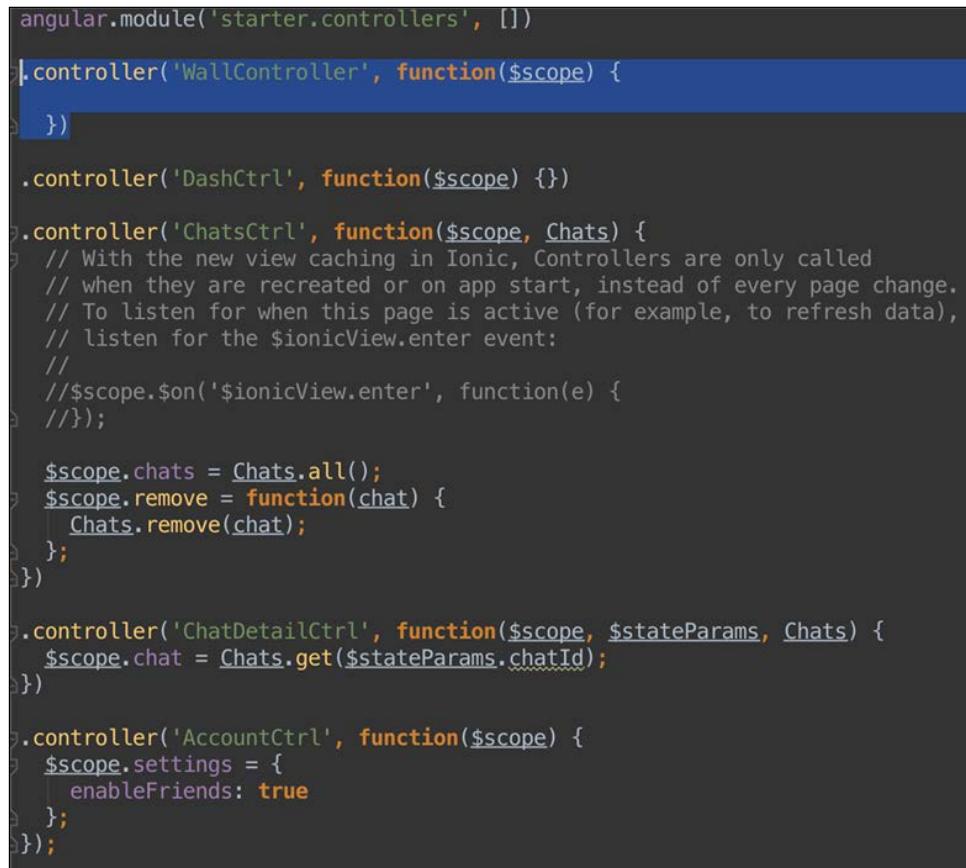
.controller('AccountCtrl', function($scope) {
  $scope.settings = {
    enableFriends: true
});
})
```

To create the `WallController` file, simply place the code found in the following code block just after the first line where you can find the line of code, `angular.module('starter.controllers', []):`

```
.controller('WallController', function($scope) {

})
```

If you have correctly replicated this code, your `controller.js` file should closely resemble to the following screenshot:



```
angular.module('starter.controllers', [])
.controller('WallController', function($scope) {
})
.controller('DashCtrl', function($scope) {})

.controller('ChatsCtrl', function($scope, Chats) {
  // With the new view caching in Ionic, Controllers are only called
  // when they are recreated or on app start, instead of every page change.
  // To listen for when this page is active (for example, to refresh data),
  // listen for the $ionicView.enter event.
  //
  // $scope.$on('$ionicView.enter', function(e) {
  //});

  $scope.chats = Chats.all();
  $scope.remove = function(chat) {
    Chats.remove(chat);
  };
})

.controller('ChatDetailCtrl', function($scope, $stateParams, Chats) {
  $scope.chat = Chats.get($stateParams.chatId);
})

.controller('AccountCtrl', function($scope) {
  $scope.settings = {
    enableFriends: true
  };
});
```

By doing this, we have successfully created the `WallController` controller. However, we still have one last step to complete the implementation of our new tab. We need to actually create the tab itself using the `<ion-tab>` element.

## Creating the tab

To create our tab, we need to revisit the `tabs.html` file. Within the file, locate the opening `<ion-tabs>` tag and place the code mentioned in the following code block just after that:

```
<!-- Wall Tab -->
<ion-tab title="Wall" icon-off="ion-ios-compose-outline" icon-on="ion-ios-compose" href="#/tab/wall">
  <ion-nav-view name="tab-wall"></ion-nav-view>
</ion-tab>
```

If you have done this correctly, your `tabs.html` file should look like what is shown in the following screenshot:



```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">
  <!-- Wall Tab -->
  <ion-tab title="Wall" icon-off="ion-ios-compose-outline" icon-on="ion-ios-compose" href="#/tab/wall">
    <ion-nav-view name="tab-wall"></ion-nav-view>
  </ion-tab>

  <!-- Dashboard Tab -->
  <ion-tab title="Status" icon-off="ion-ios-pulse" icon-on="ion-ios-pulse-strong" href="#/tab/dash">
    <ion-nav-view name="tab-dash"></ion-nav-view>
  </ion-tab>

  <!-- Chats Tab -->
  <ion-tab title="Chats" icon-off="ion-ios-chatboxes-outline" icon-on="ion-ios-chatboxes" href="#/tab/chats">
    <ion-nav-view name="tab-chats"></ion-nav-view>
  </ion-tab>

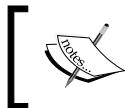
  <!-- Account Tab -->
  <ion-tab title="Account" icon-off="ion-ios-gear-outline" icon-on="ion-ios-gear" href="#/tab/account">
    <ion-nav-view name="tab-account"></ion-nav-view>
  </ion-tab>

</ion-tabs>
```

By doing this, we have successfully created a new tab in our application. Let's recap what we did to achieve this feat. First, we created a new state definition for our tab and referenced it a controller and template file. We then went ahead to create the tab itself using the `<ion-tab>` element, as in the preceding screenshot.

If you look at the preceding screenshot and pay close attention to `<ion-tab>` that we just replicated from the code block, you will see that its `<ion-nav-view>` child element has a `name` attribute with the value `tab-wall`. This is simply referencing the view we defined while defining our `tab.wall` state in our `app.js` file. These steps complete our tabs implementation.

Now, the next step is to go on and run our app and see it in action. To do this, simply run your application using the `ionic serve` technique.



To run your app using the `ionic serve` technique, simply run `ionic serve` from the root directory of your tab-app application.

If you have done this correctly, you should see something that closely resembles what we have in the following screenshots.

- For iOS:



- For Android:



## **Summary**

In this chapter, we learned about the Ionic tabs application template. We also created a tabs application called `tabs-app` and even got as far as adding a new tab of our own. In the next chapter, we will be using this same application to learn how to use Firebase to add backend services to our application.

# 9

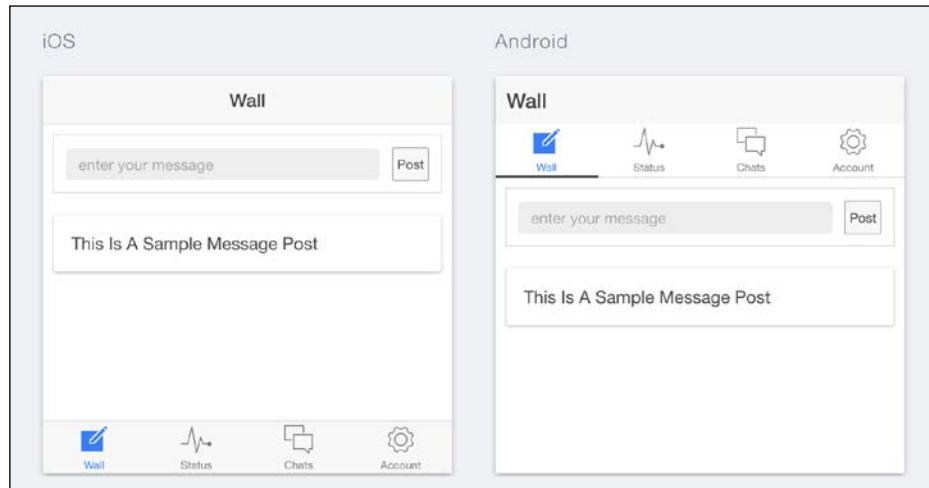
## Connecting to Firebase

In this chapter, we are going to focus solely on learning how to use Firebase to integrate a backend with our Ionic application. Firebase is a real-time data store technology that uses JSON-style database structure to let you store your data in the cloud. We will also be using the `tabs-app` app that we created in *Chapter 8, Building a Simple Social App*, to learn to integrate Firebase into our application.

### Extending our tabs-app Ionic app

In *Chapter 8, Building a Simple Social App*, we created `tabs-app`. If you recall correctly, we added a new tab called `walls`.

The basic idea we had for the `wall` tab we added was that it would be like a message board where a user could type a post and then tap the button labeled **Post** to see it on the message board, as shown in the following screenshot:



The first thing we need to do is to implement our mechanism to allow users to post, as this does not currently work in our tab-app application.

## Implementing the post wall feature

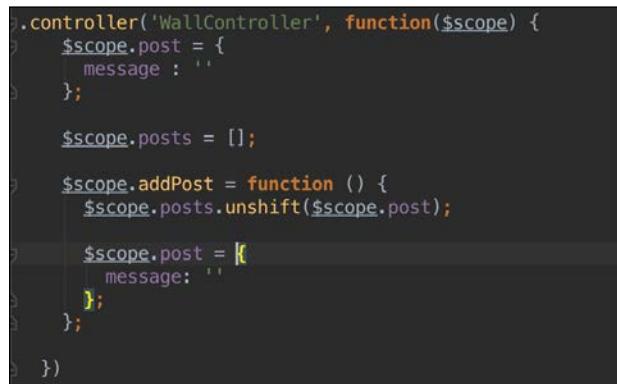
To recap what we want from our wall tab, we want to be able to enter message in the message box, as seen in the preceding screenshot, and have the message appear like the sample message post. To begin, we start by implementing the code for adding a post in our controller.

This code is provided in the following code block:

```
$scope.post = {  
    message: ''  
};  
  
$scope.posts = [];  
  
$scope.addPost = function () {  
    $scope.posts.unshift($scope.post);  
  
    $scope.post = {  
        message: ''  
    };  
};
```

You will need to replicate the code provided in the preceding block within your WallController controller. This WallController controller can be found within the controller.js file of your tabs-app application. The WallController controller can be found in the path www/js/controller.js.

If you have done this correctly, your WallController controller will look like what we have in the following screenshot:



A screenshot of a code editor showing a portion of a JavaScript file. The code defines a controller named 'WallController' with a constructor function. Inside the constructor, there is a block of code that initializes a scope variable 'post' with an empty string 'message' and an empty object '{}'. Below this, the 'posts' array is initialized as an empty array '[]'. A method 'addPost' is defined, which uses the 'unshift' array method to add the current 'post' object to the beginning of the 'posts' array. After this, the 'post' object is updated to a new one, which is an empty array '[]' with a single element 'message: '''. The code ends with a closing brace '}' for the controller's constructor.

Let's understand what this code is doing. We are simply attaching a `post` object to the controller. We are also declaring a `posts` array where all our posts will be stored.

Lastly, we have a function called `addPost()` which will add a new post to the `posts` array every time it is fired.

The next step is to wire this controller into the view of our `Wall` tab. The markup for this view is located in the `tab-wall.html` file. Now, this file looks like what we have in the following screenshot:

```
<ion-view view-title="Wall">
  <ion-content class="padding">

    <div class="list">
      <div class="item item-input-inset">
        <label class="item-input-wrapper">
          <input type="text" placeholder="enter your message">
        </label>
        <button class="button button-small">
          Post
        </button>
      </div>
    </div>

    <div class="card">
      <div class="item item-text-wrap">
        This Is A Sample Message Post
      </div>
    </div>

  </ion-content>
</ion-view>
```

You will need to completely replace the markup found within `<ion-content>` with the markup provided in the following code block:

```
<div class="list">
  <div class="item item-input-inset">
    <label class="item-input-wrapper">
      <input type="text" placeholder="enter your message"
        ng-model="post.message">
    </label>
    <button class="button button-small" on-tap="addPost() ">
      Post
    </button>
  </div>
</div>

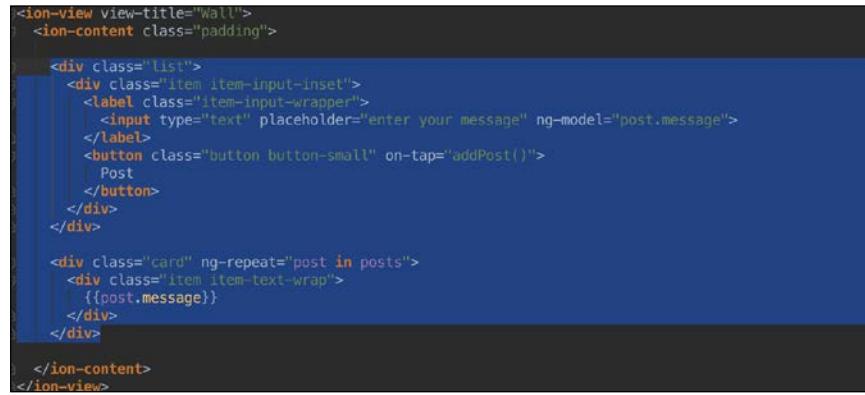
<div class="card" ng-repeat="post in posts">
  <div class="item item-text-wrap">
```

## *Connecting to Firebase*

---

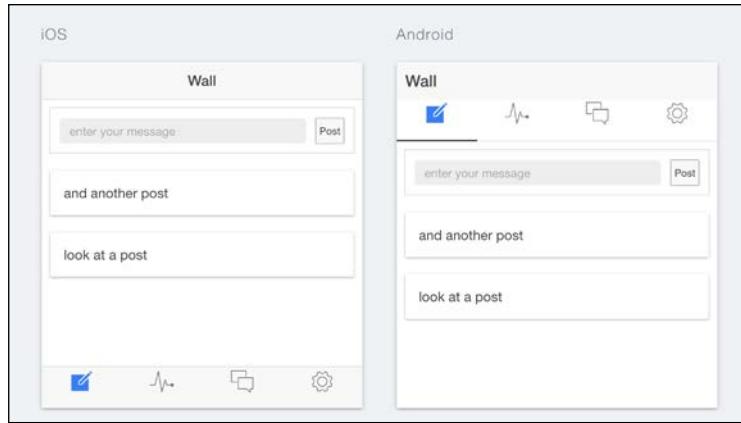
```
{ {post.message} }  
</div>  
</div>
```

If you have done this correctly, your `tab-wall.html` file will have a markup that looks like the following screenshot:



By doing this, we have completed the process of implementing and wiring our wall post feature on the Wall tab. The next step is to test it using the `ionic serve` technique. Go ahead and run your app using the `ionic serve` technique and you should see your app running in the browser.

If you try to add a message in the text box found in the Wall tab and click the **Post** button, you will see a message appear, like what we have in the following screenshot:



## The backend challenge

The one problem or challenge we have with our current application is that it does not persist. By this, we mean that once we refresh the browser, all our data is gone and we have to start again. How cool would it be if we could enter a post and when we revisited our app, we could carry on from where we left off just like every other message board in other applications? Well, we can achieve this thanks to a great technology called Firebase. The first thing we will do is try to understand Firebase and what exactly it is.

## Firebase

Before we begin this chapter, it is very important that we understand the technology we are going to be using to integrate our backend. The technology in question is called Firebase. Firebase is a technology that lets us store real-time data. Unlike traditional backend databases where you need a server running, you do not need to have a hosted server with Firebase.

All you need to get going with Firebase is an active Google account and you are good to go. Let's set up a new Firebase account.

If you do not have a Google account, you can create one by visiting <http://www.gmail.com>.

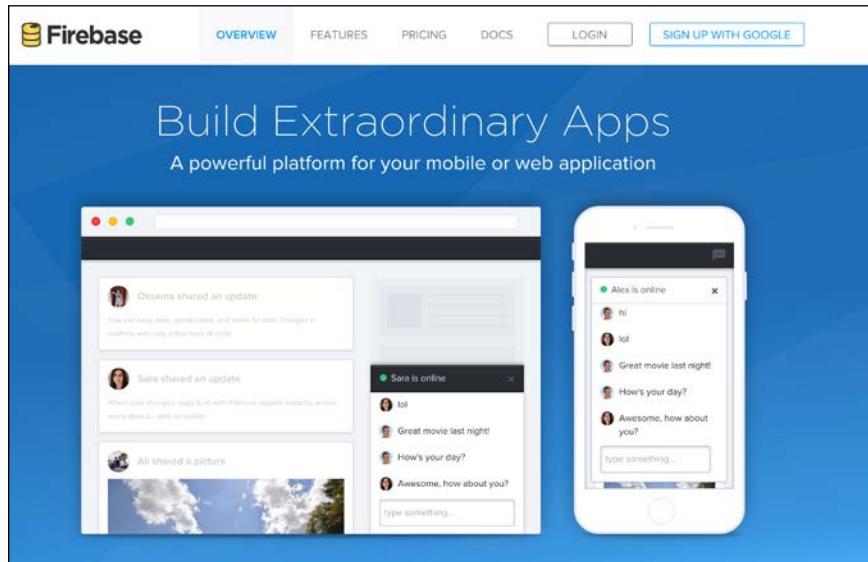
## Setting up a new Firebase account

The first thing you need to do to set up your Firebase account is go to the Firebase website, which is <http://www.firebaseio.com>.

## *Connecting to Firebase*

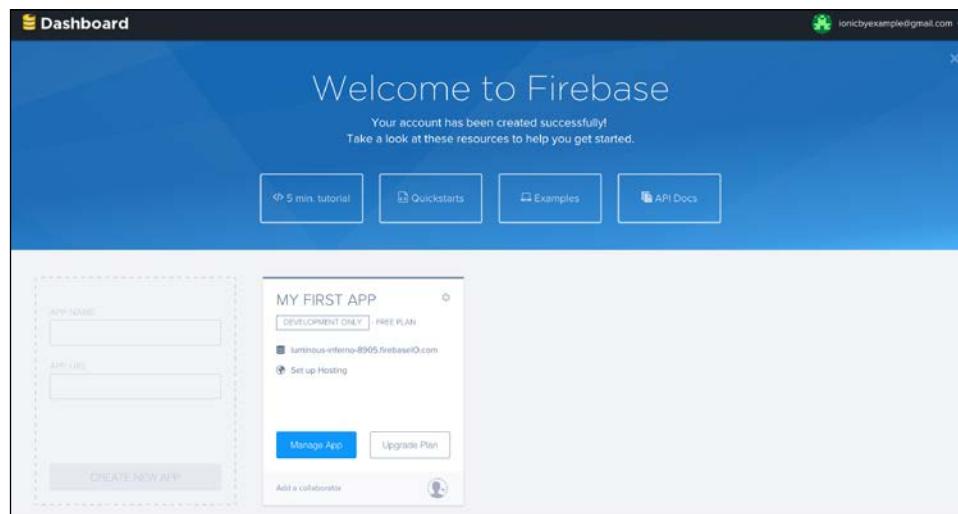
---

You should see a screen that looks like what we have in the following screenshot:



Once this is done, you should see a button labeled **Sign Up With Google** on the top right-hand corner.

When you click this button, you should see a Gmail window asking you to select or log in to a Google account. After you select the Google account you want to use, you should be redirected to your brand new Firebase account. The window you will be redirected to should look like what we have in the following screenshot:

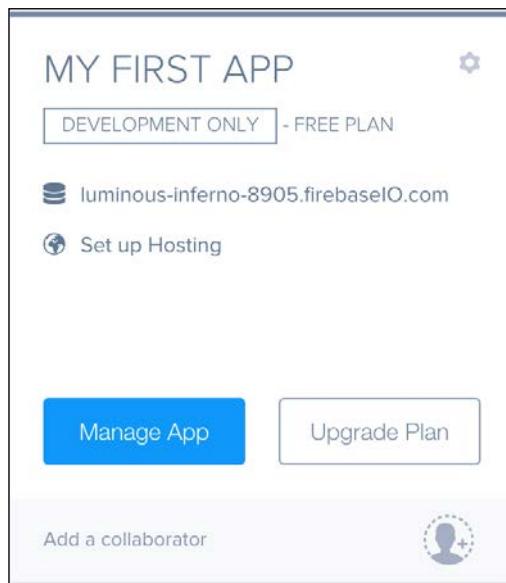




All the examples we have here are based on our sample account. You should not use the URLs from the preceding screenshot but instead use the ones you see in your own window. If you do not, your sample will not work.

You will see that there is a Firebase app created for you called **MY FIRST APP**. When using Firebase, for each app we create we also create an app for it on our Firebase dashboard. This is because Firebase uses a distinct URL to provide you access to the data of each unique application you create. So, think of this **MY FIRST APP** Firebase app as a database.

Now, let's take a closer look at **MY FIRST APP**:



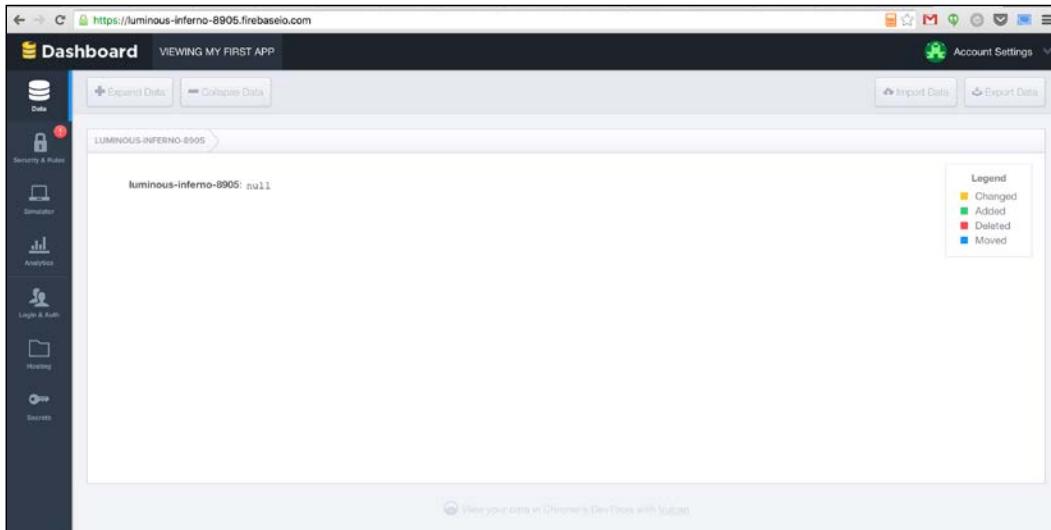
You should see something very similar to the preceding screenshot. You can access the URL for your Firebase database by clicking on the post fixed with the `.firebaseIO.com` URL. Remember that the URL you see on the screenshots will be different from the ones you see on your dashboard, and you are to use the ones on your dashboard.

You can see that the URL we have here for demonstration is `luminous-inferno-8905.firebaseio.com`.

## *Connecting to Firebase*

---

Click the URL you have on your dashboard and that should take you to your Firebase database, which should look similar to the following screenshot:



Just to clarify once again, Firebase uses URLs to access databases. What you see in the preceding screenshot is the dashboard for your database. You can also see the same database URL in the browser's address bar. Firebase uses JSON-style data structure, so basically what we send to it is JSON, and what we store is JSON too.

When we add data to our database, we will be able to see it in this dashboard.

## **Integrate Firebase into tabs-app application**

Now that we have our Firebase account and know how to get the URL of our Firebase database, the next step is to integrate it into our application.

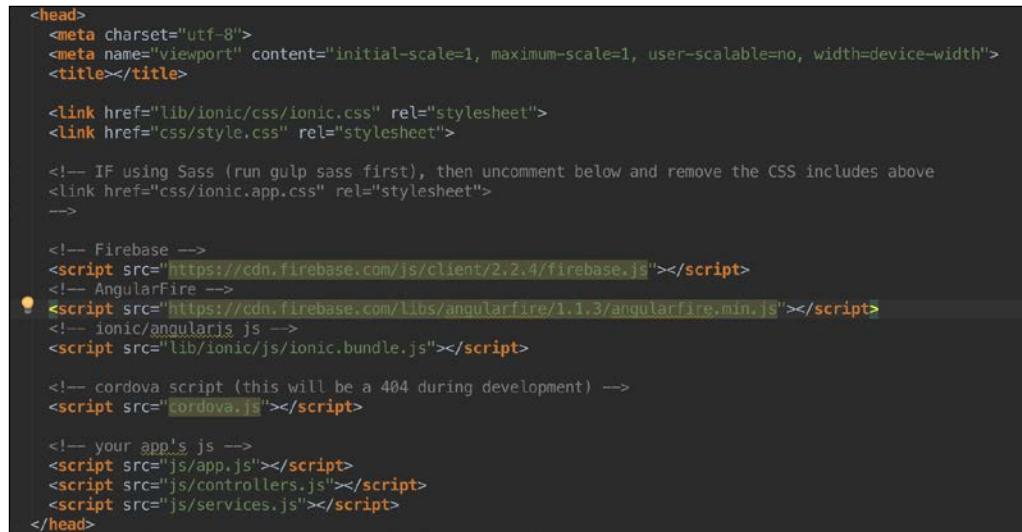
## **Adding Firebase and Angular-Fire scripts**

The first thing we need to do is to add the scripts we will need. We will need two scripts. The first is the Firebase library. The second script is the Angular-Fire library. Angular-Fire is an Angular library that makes working with Firebase in an AngularJS application much simpler.

The simplest way is to use the hosted library references. To add this to our app, open your `index.html` file and add the following script references within `<head>` of your application:

```
<!-- Firebase -->
<script src="https://cdn.firebaseio.com/js/client/2.2.4.firebaseio.js"></script>
<!-- AngularFire -->
<script src="https://cdn.firebaseio.com/libs/angularfire/1.1.3/angularfire.min.js"></script>
```

If you have done this correctly, the head part of your `index.html` should look like the following screenshot:



```
<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">

<!-- IF using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
<link href="css/ionic.app.css" rel="stylesheet">
-->

<!-- Firebase -->
<script src="https://cdn.firebaseio.com/js/client/2.2.4.firebaseio.js"></script>
<!-- AngularFire -->
● <script src="https://cdn.firebaseio.com/libs/angularfire/1.1.3/angularfire.min.js"></script>
<!-- ionic/angularjs -->
<script src="lib/ionic/js/ionic.bundle.js"></script>

<!-- cordova script (this will be a 404 during development) -->
<script src="cordova.js"></script>

<!-- your app's js -->
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/services.js"></script>
</head>
```



Make sure your references are below the Ionic bundle as seen in the preceding screenshot. This is very important or else your app will not work properly.

The next step is to reference your Angular-Fire module. This step will ensure that we can use Angular-Fire within our application. The name of this module is `firebase`. This will be added to the root module of your application, called `starter` in your `app.js` file.

Currently, this module's declaration looks something like what we have in the following screenshot:

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

You will need to add the `firebase` module as a dependent module. Doing this will make the module declaration to look something like what we have in the following screenshot:

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

You can see that the `firebase` module is now added to the module declaration as a dependency. By doing this, we have successfully integrated Firebase into the skin of our app. The next step is to actually implement it to save our data.

## Implementing Firebase to our app

To implement Firebase in our app, we will need to do some work within our `WallController` controller. The first thing we need to code for is the ability to pull items from the database. The second thing we need to code for is the ability to add items to the database.

## Pulling from database

The first thing we need to do is to add the `$firebaseArray` service dependency into our `WallController` controller. This service is part of the Angular-Fire library and makes it easy for us to work with arrays in Firebase.

Adding the service dependency correctly should make your `WallController` controller definition look like what we have in the following screenshot:

```
.controller('WallController', function($scope, $firebaseArray) {
```

The next step is to actually write code to pull the data from the database. Replicate the code provided in the following code block in your `WallController` controller:

```
var postsDatabaseRef = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com").child('posts');
var postsData = $firebaseArray(postsDatabaseRef);
```

This piece of code creates a new Firebase reference at first. We passed in the URL of the Firebase database that we created earlier. Make sure you change the placeholder text (YOUR-FIREBASE-APP) to reflect the URL of your Firebase database.

After this, we used the \$firebase service that we added earlier to create a path called postData. The last step we need to do is to allow our app to load data from this postData path and use it. To do this, we need to edit the code of our WallController slightly. Currently, our WallController controller's code looks like what we have in the following screenshot:

```
.controller('WallController', function($scope, $firebaseArray) {
  var postsDatabaseRef = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com");
  var postData = $firebaseArray(postsDatabaseRef).child('posts');

  $scope.post = {
    message: ''
  };

  $scope.posts = [];

  $scope.addPost = function () {
    $scope.posts.unshift($scope.post);

    $scope.post = {
      message: ''
    };
  };
})
```

Pay close attention to the piece of code highlighted in the preceding screenshot. We need to edit this piece of code such that instead of equating to an empty array, it should equate to our postData variable. Doing this correctly should make us end up with a WallController controller that looks like the following screenshot:

```
.controller('WallController', function($scope, $firebaseArray) {
  var postsDatabaseRef = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com");
  var postData = $firebaseArray(postsDatabaseRef).child('posts');

  $scope.post = {
    message: ''
  };

  $scope.posts = postData;

  $scope.addPost = function () {
    $scope.posts.unshift($scope.post);

    $scope.post = {
      message: ''
    };
  };
})
```

By doing this, we have implemented the first part; our Firebase implementation and our app now loads data from our database. The next step is to implement the code to add our posts to our database.

## Adding to database

Adding to the database is actually pretty easy. All we need to do is slightly edit our `addPost()` function. Currently, our `addPost()` function looks like what we have in the following screenshot:



```
$scope.addPost = function () {
  $scope.posts.unshift($scope.post);

  $scope.post = {
    message: ''
  };
}
```

To make our data persist in our database, we only need to replace the code highlighted in the preceding screenshot with the following code block:

```
$scope.posts.$add($scope.post);
```

Now, your `addPost()` function should look like what we have in the following screenshot:



```
$scope.addPost = function () {
  $scope.posts.$add($scope.post);

  $scope.post = {
    message: ''
  };
}
```

All we did was just change the `unshift()` method to the `$add()` method. The `$add()` method is a method from Firebase that adds items to a Firebase database. At this point, we have completed the implementation of our backend. As easy as that was, we have a working database in just a few short steps and can now test this live. Your final `WallController` controller should look like the following code block:

```
.controller('WallController', function($scope, $firebaseArray) {
  var postsDatabaseRef = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com").child('posts');
  var postsData = $firebaseArray(postsDatabaseRef);

  $scope.post = {
```

```
        message : ''  
    };  
  
    $scope.posts = postsData;  
  
    $scope.addPost = function () {  
        $scope.posts.$add($scope.post);  
  
        $scope.post = {  
            message: ''  
        };  
    };  
})
```

To test your application, simply run your app using the `ionic serve` technique. When you do this, you should be able to enter messages in your application, and even after you refresh your browser, the data that you have already posted will still exist. Also, if you have a look at the Firebase dashboard for your database, you will see that the data you entered in the app is present there.

## Summary

In this chapter, we learned some really cool ways of using Firebase to easily add a backend to our Ionic app. We only touched upon what Firebase lets us do, and you can look at the Firebase documentation available at <https://www.firebaseio.com/docs/> to see the full features of Firebase.

At this point, we have almost come to the end of our book. The next chapter will be the final one, and it is one you should definitely read. It contains some very useful information on how to harness skills learned in this book to get even better at using Ionic.



# 10

## Roundup

In this chapter, we are going to have an overview of the important things that we haven't covered yet about Ionic and which you might find very useful. You will also learn some useful tips about Ionic and discover some great tips about how to make even better use of Ionic to develop great apps.

### Uncovered features of Ionic

Although we covered many great topics in this book, there are a lot of great features that we did not cover as they were beyond the scope of this book. We mostly focused on the core features of Ionic, such as how to get Ionic set up. We then learned to create Ionic apps using the blank, side menu, and tabs templates. We also learned to test our Ionic application using the `ionic serve` technique.

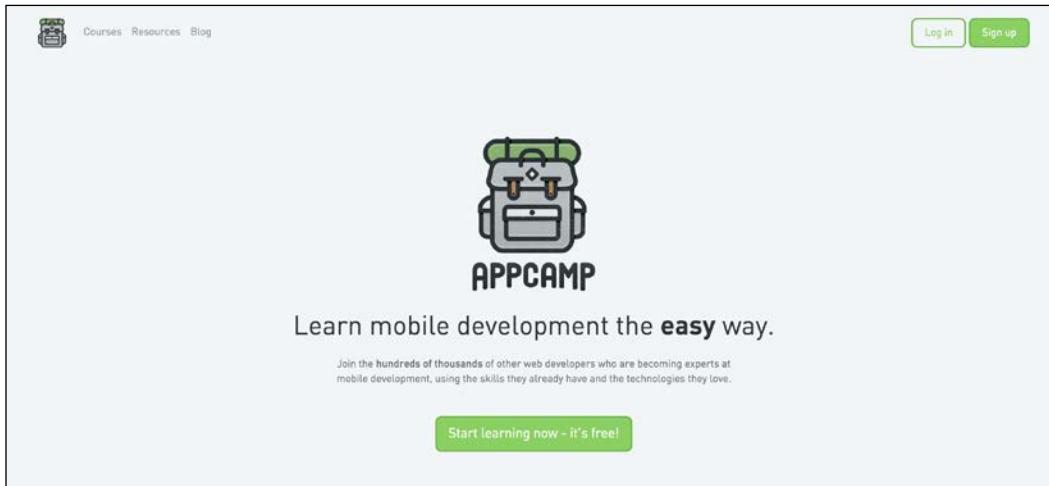
In this section of the book, I will name a couple of things that will be useful for you to get to grips with in order to become better at Ionic.

### Appcamp.IO

[Appcamp . IO](http://appcamp.io) is a free website created by some of the Ionic staff. It is a place where you can go and learn some tips and tricks that will sharpen your Ionic development skills.

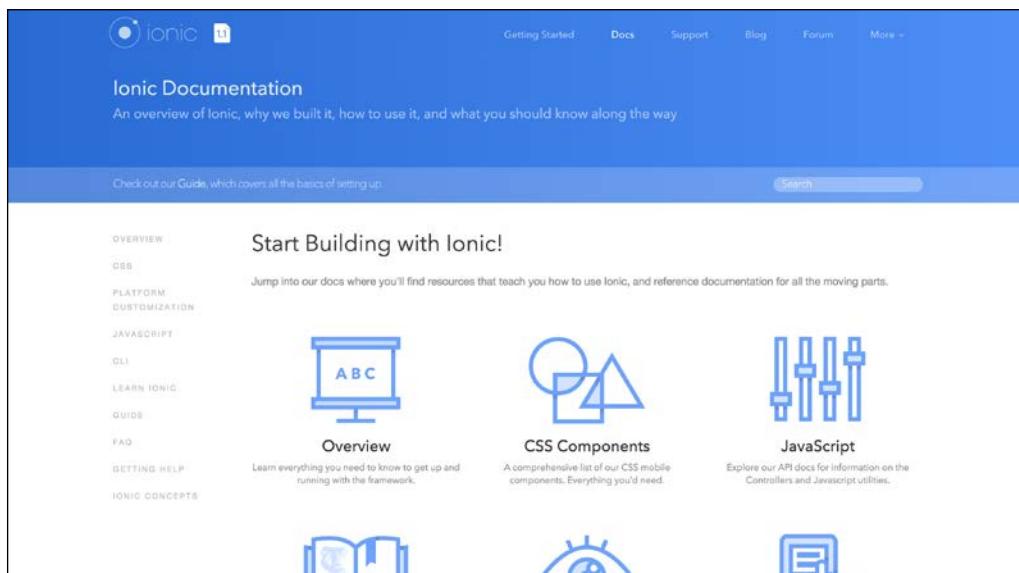
## *Roundup*

The content on Appcamp.io is great for beginners, and it is in some ways in line with the philosophy of this book.



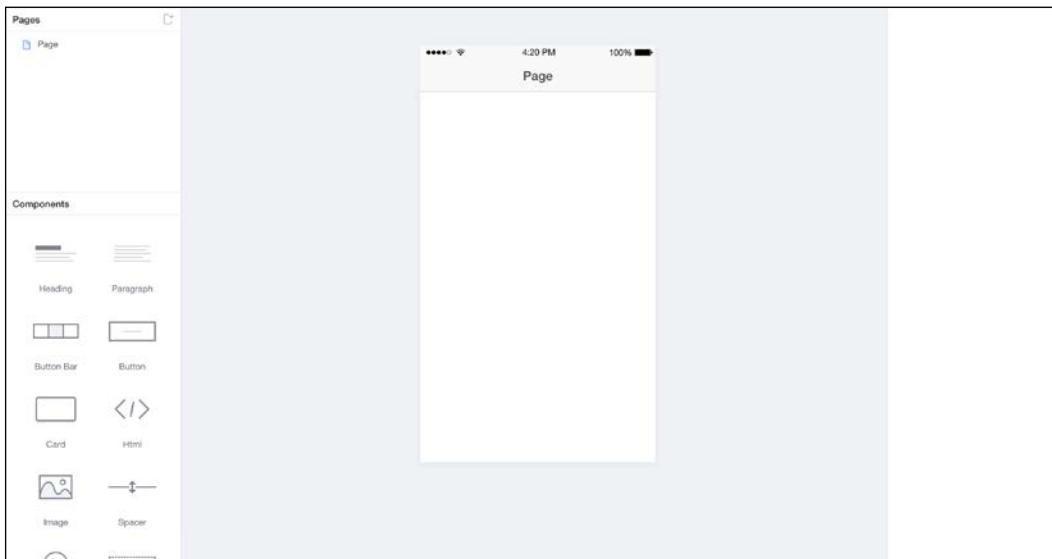
## **The Ionic documentation**

The Ionic documentation page is pretty much the Bible for everything on Ionic. Ionic is very well documented and any feature you want to use is provided there with the sample code and how to use it. You can access the Ionic documentation at <http://www.ionicframework.com/docs>:



## The Ionic creator

This is the drag-and-drop tool built by Ionic for people who want to design their first app or people with limited coding skills. Its greatest feature is that anything you design by dragging and dropping Ionic elements can be tested in the browser, and the code can be extracted as a ready-to-go application. This is great news for designers who don't know how to code as they can quickly use the visual drag-and-drop features of the Ionic creator to design their apps and pass on the code to seasoned developers. You can visit the Ionic creator website at <https://creator.ionic.io>.



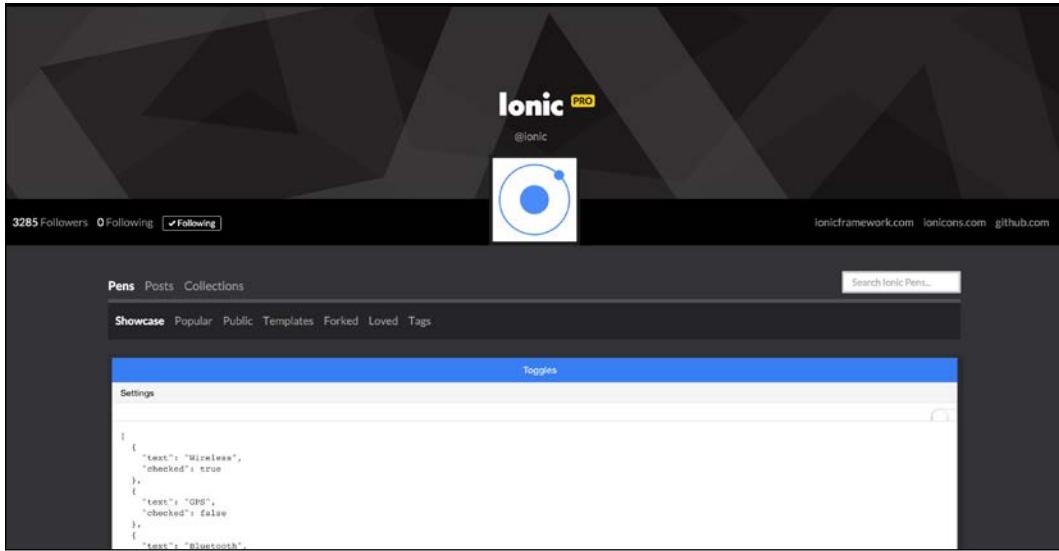
## The Ionic code pen

Sometimes, even after you have visited the documentation for some component of Ionic, you will still want to see a real code sample. Or maybe you want to try to reproduce a bug to show others. This is where the Ionic code pen site shines. It is a place where you can find some really great implementations of different features with the code available for you to learn from. It is also the best way to showcase a bug to people who can see it and help you resolve any issues.

## *Roundup*

---

You can visit the Ionic code pen website at <http://codepen.io/ionic>.

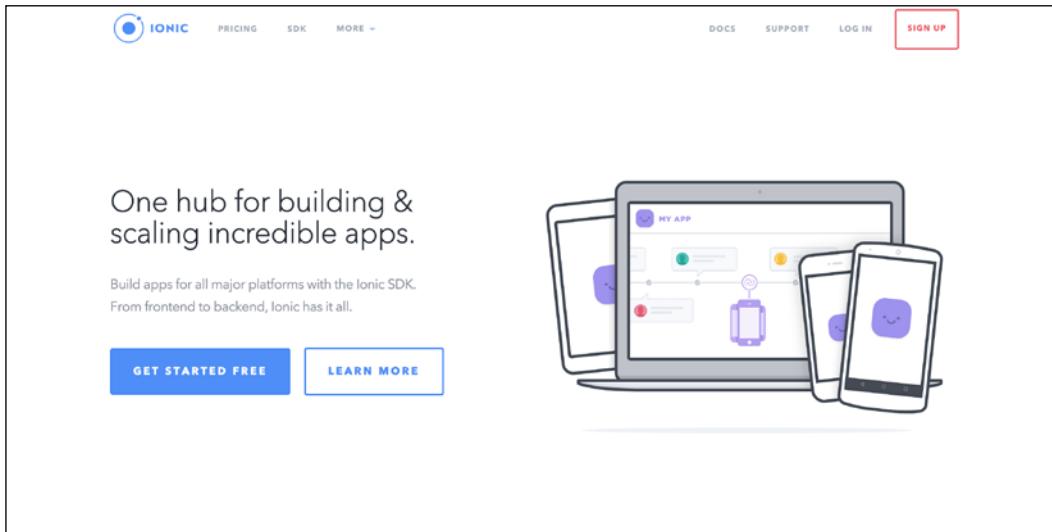


## Ionic.IO

The Ionic.IO platform is a complete suite of premium tools that enable you to add great services to your Ionic application. With Ionic.IO, you can add things like **Push Notification**, **Analytics**, and the ability to build your app for the app store in the cloud. There is also Ionic deploy, which is a feature that lets you update your app live without resubmitting it to the app store.

At the time of writing this book, the Ionic.IO tools were all in beta, and although they were free at the time of writing, Ionic has announced that they will be paid services in future. This is something that you should closely follow as you might find yourself needing to use some of the services provided by the Ionic.IO platform.

You can visit the Ionic.IO platform at <http://www.ionic.io>.



## The Ionic playground

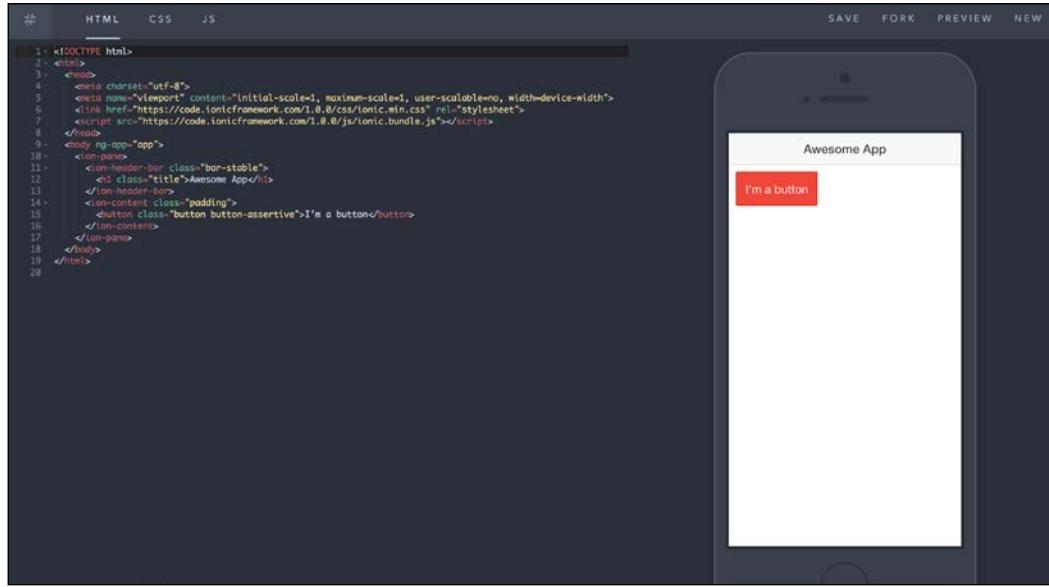
The Ionic playground is a very new and useful tool to quickly bootstrap an idea. It is a great way to simply spin off code and have it show you the results right there in the browser. I personally see this tool as very useful in the educational world as it makes it easy to create Ionic applications with only a browser.

If you find yourself needing to showcase/demo something quickly without any setup, and have a computer close-by, then make sure you give the Ionic playground a try.

## Roundup

---

You can visit the Ionic playground at <http://play.ionic.io/>.



## The Ionic community

One of the chief reasons Ionic is very successful is the fact that it has a strong active community. From social media and meetup groups to open source activists, Ionic has a wide range of support from people around the world. This means that if you do run into problems of any sort, you are never too far from help. With that in mind, here are some great links to community elements that you might want to keep an eye on:

- **The Ionic forum:** The Ionic forum really is a great place to voice your troubles or find solutions to shared problems. There are multiple active members and users who also get motivational badges for contributing to the forum by commenting and helping others find solutions. You can access this forum at <http://forum.ionicframework.com>.
- **The Ionic Slack channel:** This channel has over 4000 people active on it 24/7 and in multiple countries. It is a great place to meet people, find work, look for help, employ someone, or just simply express yourself. To join the Ionic Slack channel, simply request an invitation for free at <http://ionicworldwide.herokuapp.com>.
- **Twitter:** Twitter is the most vibrant social medium when it comes to finding the latest and greatest thing about Ionic. If you use Twitter, you can follow @ionicframework for more information and updates.

- **The Ionic blog:** Ionic writes blog posts very actively about various topics relating to using Ionic. Some of these posts could be about critical updates, inspirational stories, or even announcements of new products or features. You can find the Ionic blog at <http://blog.ionic.io>.
- **Ionic meetup groups:** Around the world, there are hundreds of meetup groups autonomously hosting events and meetups. This is a community effort by local people to grow Ionic awareness in their locality, and you are almost guaranteed to have one in your nearby city. If you do not have one, feel free to start your own local meetup. You can find a list of all meetups at <http://blog.ionic.io/ionic-worldwide>.

The community around Ionic is pretty much the main reason why it grew so rapidly, and you should be sure to trust the community for any needs. As a note, make sure to use the skills you have learned from this book to really strive and improve your Ionic skills and build some great mobile applications. Remember that nothing is too simple to be great and nothing is too great to be too difficult to build.

## Useful resources

The following are some useful links to some sites and resources that will aid you further in your quest to learn more about Ionic:

- **The Ionic framework:** <http://www.ionicframework.com>
- **The Ionic GitHub:** <http://www.github.com/driftyco/ionic>
- **AngularJS:** <http://www.angularjs.org>
- **Ionic stack overflow:** <http://stackoverflow.com/questions/tagged/ionic-framework>
- **Firebase:** <http://www.firebaseio.com>
- **NodeJS:** <http://www.nodejs.org>
- **Bower:** <http://www.bower.io>
- **Gulp:** <http://www.gulpjs.com>
- **Cordova:** <https://cordova.apache.org>
- **Ionic market:** <https://market.ionic.io>
- **ngCordova:** <http://ngcordova.com>
- **Ionic jobs:** <http://jobs.ionic.io>
- **Ionic showcase:** <http://showcase.ionicframework.com>
- **Ionic lab:** <http://lab.ionic.io>

## **Summary**

This chapter was a roundup of Ionic and all its features. I hope you will now know how to build rich features for your mobile applications and have them possess native-like features with the help of Ionic.

# Module 3

## Ionic Cookbook

Over 35 exciting recipes to spice up your application development with Ionic



# 1

## Creating Our First App with Ionic

In this chapter, we will cover:

- ▶ Setting up a development environment
- ▶ Creating a HelloWorld app via CLI
- ▶ Creating a HelloWorld app via Ionic Creator
- ▶ Copying examples from Ionic Codepen Demos
- ▶ Viewing the app using your web browser
- ▶ Viewing the app using iOS Simulator
- ▶ Viewing the app using Xcode for iOS
- ▶ Viewing the app using Genymotion for Android
- ▶ Viewing the app using Ionic View
- ▶ Customizing the app folder structure

### Introduction

There are many options for developing mobile applications today. Native applications require a unique implementation for each platform, such as iOS, Android, and Windows Phone. It's required for some use cases such as high-performance CPU and GPU processing with lots of memory consumption. Any application that does not need over-the-top graphics and intensive CPU processing could benefit greatly from a cost-effective, write once, and run everywhere HTML5 mobile implementation.

For those who choose the HTML5 route, there are many great choices in this active market. Some options may be very easy to start but could be very hard to scale or could face performance problems. Commercial options are generally expensive for small developers to discover product and market fit. It's a best practice to think of the users first. There are instances where a simple responsive design website is a better choice; for example, the business has mainly fixed content with minimal updating required or the content is better off on the web for SEO purposes.

Ionic has several advantages over its competitors:

- ▶ It's written on top of AngularJS
- ▶ UI performance is strong because of its use of the `requestAnimationFrame()` technique
- ▶ It offers a beautiful and comprehensive set of default styles, similar to a mobile-focused Twitter Bootstrap
- ▶ Sass is available for quick, easy, and effective theme customization

In this chapter, you will go through several HelloWorld examples to bootstrap your Ionic app. This process will give you a quick skeleton to start building more comprehensive apps. The majority of apps have similar user experience flows such as tabs and a side menu.

## Setting up a development environment

Before you create the first app, your environment must have the required components ready. Those components ensure a smooth process of development, build, and test. The default Ionic project folder is based on Cordova's. Therefore you will need the Ionic CLI to automatically add the correct platform (that is, iOS, Android, or Windows Phone) and build the project. This will ensure all Cordova plugins are included properly. The tool has many options to run your app in the browser or simulator with live reload.

### Getting ready

You need to install Ionic and its dependencies to get started. Ionic itself is just a collection of CSS styles and AngularJS Directives and Services. It also has a command-line tool to help manage all of the technologies such as Cordova and Bower. The installation process will give you a command line to generate initial code and build the app.

Ionic uses npm as the installer, which is included when installing Node.js. Please install the latest version of Node.js from <http://nodejs.org/download/>.

You will need Cordova, `ios-sim` (iOS Simulator), and Ionic:

```
$ npm install -g cordova ionic ios-sim
```

This single command line will install all three components instead of issuing three command lines separately. The `-g` parameter is to install the package globally (not just in the current directory).

For Linux and Mac, you may need to use the `sudo` command to allow system access:

```
$ sudo npm install -g cordova ionic ios-sim
```

There are a few common options for an integrated development environment:

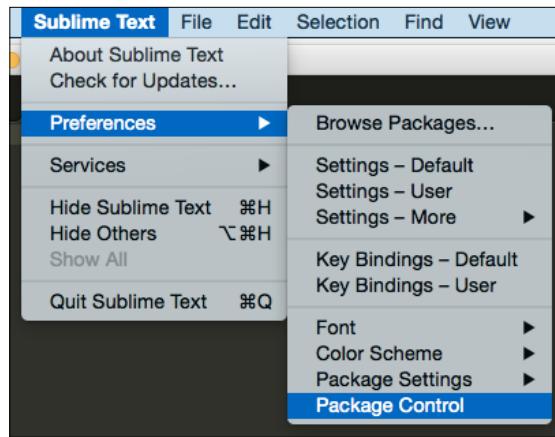
- ▶ Xcode for iOS
- ▶ Eclipse or Android Studio for Android
- ▶ Microsoft Visual Studio Express or Visual Studio for Windows Phone
- ▶ Sublime Text (<http://www.sublimetext.com/>) for web development

All of those have a free license. Sublime Text is free for non-commercial use only but you have to purchase a license if you are a commercial developer. Most frontend developers would prefer to use Sublime Text for coding HTML and JavaScript because it's very lightweight and comes with a well-supported developer community. You could code directly in Xcode, Eclipse, or Visual Studio Express, but those are somewhat *heavy duty* for web apps, especially when you have a lot of windows open and just need something simple to code.

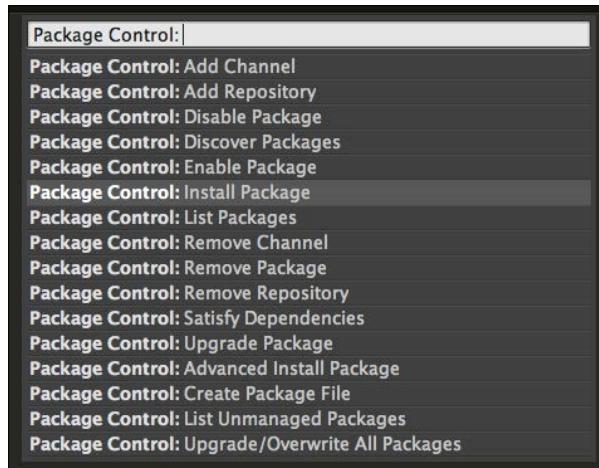
## How to do it...

If you decide to use Sublime Text, you will need Package Control (<https://packagecontrol.io/installation>), which is similar to a **Plugin Manager**. Since Ionic uses Sass, it's optional to install the Sass Syntax Highlighting package:

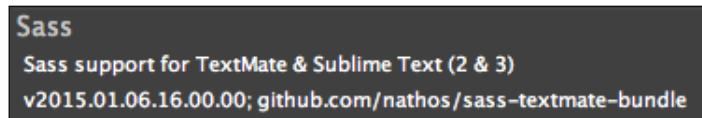
1. Select **Sublime Text | Preferences | Package Control**:



2. Select **Package Control: Install Package**. You could also just type the commands partially (that is, `inst`) and it will automatically select the right option.



3. Type `Sass` and the search results will show one option for **TextMate & Sublime Text**. Select that item to install.



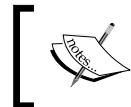
## See also

There are tons of packages that you may want to use, such as Haml, JSHint, JSLint, Tag, ColorPicker, and so on. You can browse around this website: <https://sublime.wbond.net/browse/popular>, for more information.

## Creating a HelloWorld app via CLI

It's quickest to start your app using existing templates. Ionic gives you three standard templates out of the box via the command line:

- ▶ **Blank**: This template has a simple one page with minimal JavaScript code.
- ▶ **Tabs**: This template has multiple pages with routes. A route URL goes to one tab or tabs.
- ▶ **Sidemenu**: This is template with the left and/or right menu and with center content area.



There are two other additional templates: maps and salesforce. But these are very specific to apps using Google Maps or for integration with the Salesforce.com API.



## How to do it...

To set up the app with a blank template from Ionic, use this command:

```
$ ionic start HelloWorld_Bank blank
```



If you don't have an account in <http://ionic.io/>, the command line will ask for it. You could either press **y** or **n** to continue. It's not required to have an account at this step.



If you replace `blank` with `tabs`, it will create a tab template:

```
$ ionic start HelloWorld_Tabs tabs
```

Similarly, this command will create an app with a sidemenu:

```
$ ionic start HelloWorld_Sidemenu sidemenu
```

The sidemenu template is the most common template as it provides a very nice routing example with different pages in the `templates` folder under `/www`.

Additional guidance for the Ionic CLI is available on the GitHub page:

<https://github.com/driftyco/ionic-cli>

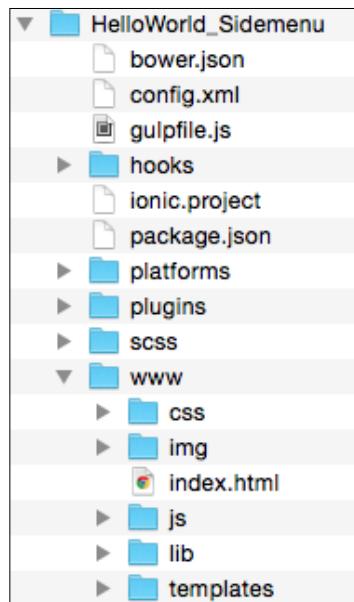
## How it works...

This chapter will show you how to quickly start your codebase and visually see the result. More detail about AngularJS and its template structure will be discussed across various chapters in this book. However, the following are the core concepts:

- ▶ **Controller:** Manage variables and models in the scope and trigger others, such as services or states.
- ▶ **Directive:** Where you manipulate the DOM, since the directive is bound to a DOM object.

- ▶ **Service:** Abstraction to manage models or collections of complex logic beside get/set required.
- ▶ **Filter:** Mainly used to process an expression in the template and return some data (that is, rounding number, add currency) by using the format `{ { expression | filter } }`. For example, `{ { amount | currency } }` will return \$100 if the amount variable is 100.

The project folder structure will look like the following:



You will spend most of your time in the /www folder, because that's where your application logic and views will be placed.

By default from the Ionic template, the AngularJS module name is called `starter`. You will see something like this in `app.js`, which is the bootstrap file for the entire app:

```
angular.module('starter', ['ionic', 'ngCordova',
  'starter.controllers', 'starter.services', 'starter.directives',
  'starter.filters'])
```

This basically declares `starter` to be included in `ng-app="starter"` of `index.html`. We would always have `ionic` and `ngCordova` (as in other examples from this book, although `ngCordova` is not essential). The other modules are required and listed in the array of string [...] as well. They can be defined in separate files.

Note that if you double click on the `index.html` file to open in the browser, it will show a blank page. This doesn't mean the app isn't working. The reason is that the AngularJS component of Ionic dynamically loads all the `.js` files and this behavior requires server access via an HTTP protocol (`http://`). If you open a file locally, the browser automatically treats it as a file protocol (`file://`) and therefore AngularJS will not have the ability to load additional `.js` modules to run the app properly. There are several methods of running the app that will be discussed.

## Creating a HelloWorld app via Ionic Creator

Another way to start your app codebase is to use Ionic Creator. This is a great interface builder to accelerate your app development with a drag-and-drop style. You can quickly take existing components and position them to visualize how it should look in the app via a web-based interface. Most common components like buttons, images, checkboxes, and so on are available.

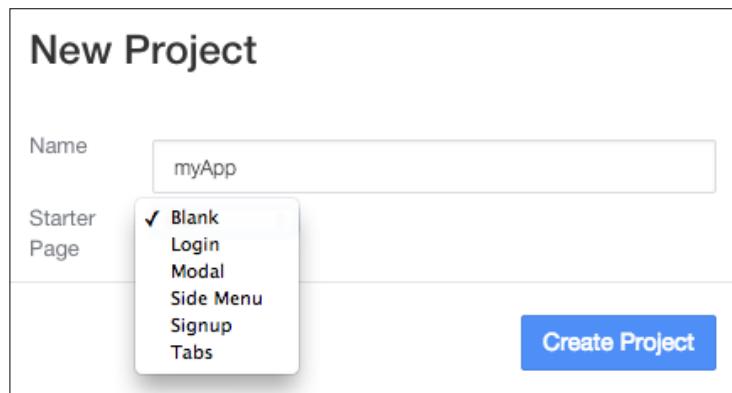
Ionic Creator allows the user to export everything as a project with all `.html`, `.css`, and `.js` files. You should be able edit content in the `/www` folder to build on top of the interface.

### Getting ready

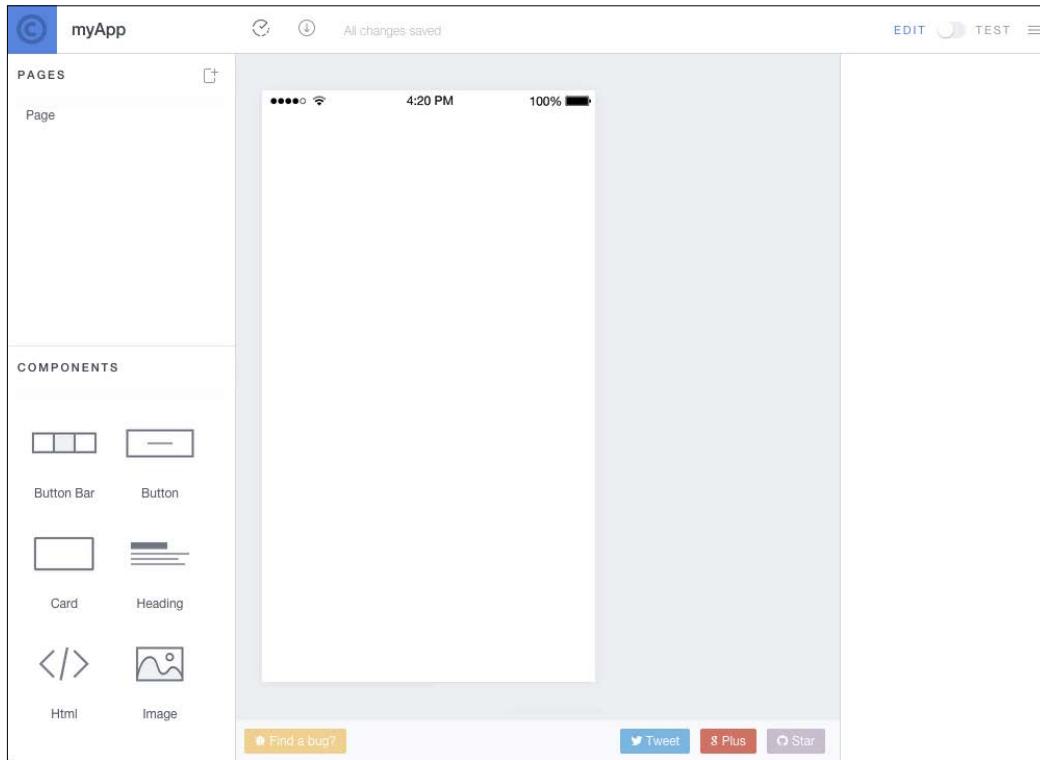
Ionic Creator requires registration for a free account at <https://creator.ionic.io/> to get started.

### How to do it...

Create a new project called `myApp`:



You will see this simple screen:



The center area is your app interface. The left side gives you a list of *pages*. Each page is a single route. You also have access to a number of UI components that you would normally have to code by hand in an HTML file. The right panel shows the properties of any selected component.

You're free to do whatever you need to do here by dropping components to the center screen. If you need to create a new page, you have to click the plus sign in the **Pages** panel. Each page is represented as a *link*, which is basically a route in AngularJS UI Router's definition. To navigate to another page (for example, after clicking a button), you can just change the **Link** property and point to that page.

There is an **Edit** button on top where you can toggle back and forth between Edit Mode and Preview Mode. It's very useful to see how your app will look and behave.

Once completed, click on the **Export** button on the top navigation. You have three options:

- ▶ Use the Ionic CLI tool to get the code
- ▶ Download the project as a zip file
- ▶ Review the raw HTML

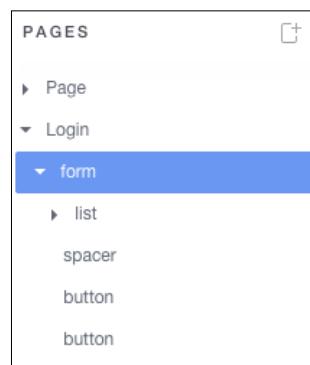
## Export

Ionic CLI   ZIP File   Raw HTML

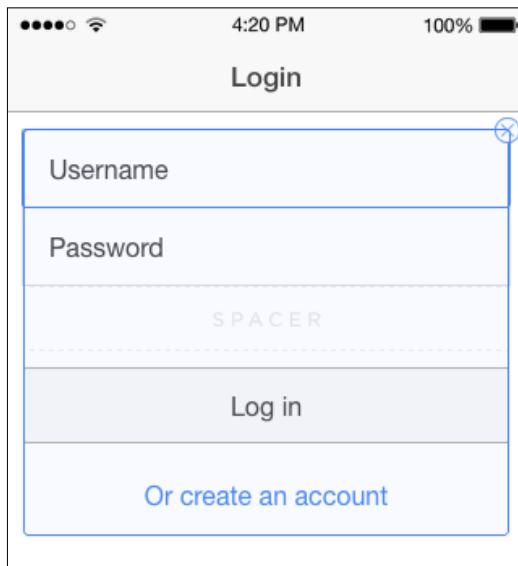
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no,
width=device-width">
6     <title></title>
7
8   <style>
9     .angular-google-map-container {
10       width: 100%;
11       height: 504px;
12     }
13   </style>
14
15   <link href="/css/preview-frame.css" rel="stylesheet">
16   <link href="/lib/ionic.css" rel="stylesheet">
17
18   <!-- ionic/angularjs is -->
```

Done

The best way to learn Ionic Creator is to play with it. You can add a new page and pick out any existing templates. This example shows a Login page template:



Here is how it should look out of the box:



### There's more...

To switch to Preview Mode where you can see the UI in a device simulator, click the switch button on the top right to enable **Test**:



In this mode, you should be able to interact with the components in the web browser as if it's actually deployed on the device.

If you break something, it's very simple to start a new project. It's a great tool to use for "prototyping" and to get initial template or project scaffolding. You should continue to code in your regular IDE for the rest of the app. Ionic Creator doesn't do everything for you, yet. For example, if you want to access specific Cordova plugin features, you have to write that code separately.

Also, if you want to tweak the interface outside of what is allowed within Ionic Creator, it will also require specific modifications to the `.html` and `.css` files.

## Copying examples from Ionic Codepen Demos

Sometimes it's easier to just get snippets of code from the example library. Ionic Codepen Demos (<http://codepen.io/ionic/public-list/>) is a great website to visit. Codepen.io is a playground (or sandbox) to demonstrate and learn web development. There are other alternatives such as [plnkr.com](http://plnkr.com) or [jsfiddle.com](http://jsfiddle.com). It's just a developer's personal preference which one to choose.

However, all Ionic's demos are already available on Codepen, where you can experiment and clone to your own account. <http://plnkr.com> has an existing AngularJS boilerplate and could be used to just practice specific AngularJS areas because you can copy the link of sample code and post on Stackoverflow.com if you have questions.

### How to do it...

There are several tags of interest to browse through if you want specific UI component examples:

ionic	33	angularjs	26	tabs	10
mobile	7	nav-view	6	menus	4
list	4	starter	3	swipe	3
svg	2	modal	2	icon	2
header	2	pull to refresh	2	search	2
popup	2	dialog	2	navigation	2
google maps	1	sign-in	1	map	1
css3	1	accordion	1	radio	1
css animations	1	split-view	1	text-clip	1
contrib	1	toggle	1	chrome	1
frosted	1	checkbox	1	thumbnails	1
html5	1	tinder	1	card	1
frosted glass	1	pop-up	1	chat	1
images	1	flickr	1	resource	1

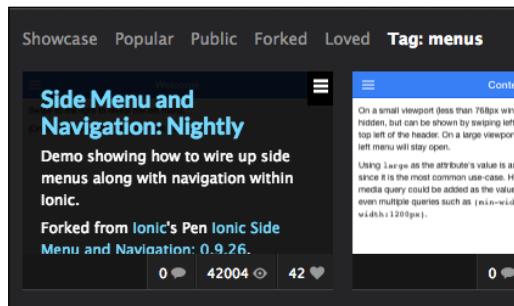
You don't need a Codepen account to view. However, if there is a need to save a custom **pen** and share with others, free registration will be required.

The Ionic Codepen Demos site has more collections of demos comparing to the CLI. Some are based on a nightly build of the platform so they could be unstable to use.

## There's more...

You can find the same side menu example on this site:

1. Navigate to <http://codepen.io/ionic/public-list/> from your browser.
2. Select **Tag: menus** and then click on **Side Menu and Navigation: Nightly**.



3. Change the layout to fit a proper mobile screen by clicking on the first icon of the layout icons row on the bottom right of the screen.

A screenshot of the CodePen editor interface. The top bar shows the CodePen logo and sharing options. The main area is divided into three tabs: HTML, CSS, and JS. The HTML tab contains the Ionic framework code for the side menu. The CSS tab contains a simple body style with a cursor icon. The JS tab contains the AngularJS configuration for the application. On the right, a preview window shows a mobile application with a left-side menu and a "Welcome" page. A tooltip at the top right of the preview area says "Swipe to the right to reveal the left menu. (On desktop click and drag from left to right)". The bottom of the editor shows tabs for Collections, Embed, Details, and a search bar, along with buttons for Pen Settings, HTML, CSS, and JS.

## Viewing the app using your web browser

In order to "run" the web app, you need to turn your /www folder into a web server. Again there are many methods to do this and people tend to stick with one or two ways to keep things simple. A few other options are unreliable such as Sublime Text's live watch package or static page generator (for example, Jekyll, Middleman App, and so on). They are slow to detect changes and may freeze your IDE so these won't be mentioned here.

### Getting ready

The recommended method is to use the `ionic serve` command line. It basically launches an HTTP server so you can open your app in a desktop browser.

### How to do it...

1. First you need to be in the project folder. Let's assume it is the Side Menu HelloWorld:

```
$ cd HelloWorld_Sidemenu
```

2. From there, just issue the simple command line:

```
$ ionic serve
```

That's it! There is no need to go into the /www folder or figure out which port to use. The command line will provide these options while the web server is running:

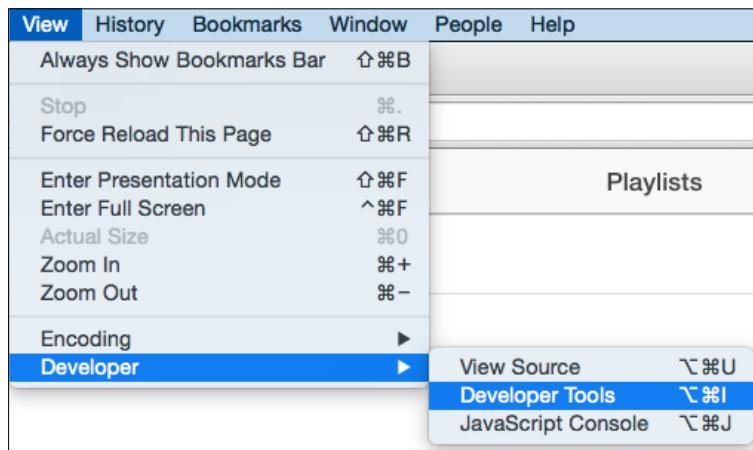
```
Running dev server: http://localhost:8100
Running live reload server: http://localhost:35729
Watching : [ 'www/**/*', '!www/lib/**/*' ]
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit
```

The most common option to use here is `r` to restart or `q` to quit when you are done.

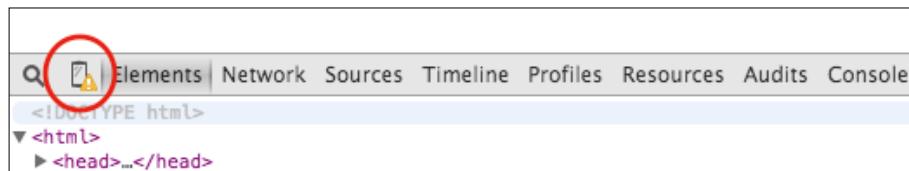
There is an additional step to view the app with the correct device resolution:

1. Install Google Chrome if it's not already on your computer.
2. Open the link (for example, `http://localhost:8100/#/app/playlists`) from `ionic serve` in Google Chrome.

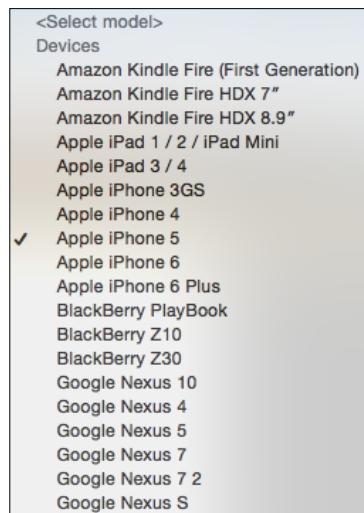
3. Turn on Developer Tools. For example, in Mac's Google Chrome, select **View | Developer | Developer Tools**:



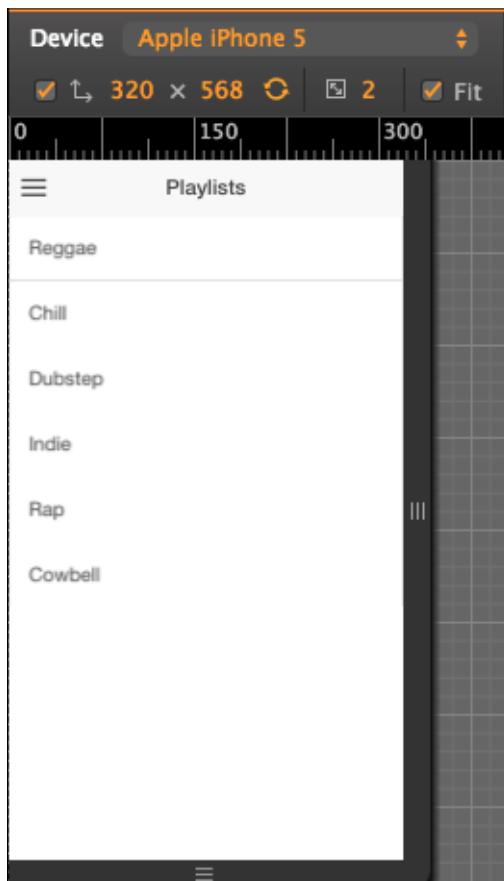
4. Click on the small mobile icon in the Chrome Developer Tools area:



5. There will be a long list of devices to pick from:



- After selecting a device, you need to refresh the page to ensure the UI is updated. Chrome should give you the exact view resolution of the device.



Most developers would prefer to use this method to code as you can debug the app using Chrome Developer Tools. It works exactly like any web application. You can create breakpoints or output variables to the console.

### How it works...

Note that `ionic serve` is actually watching everything under the `/www` folder except the JavaScript modules in the `/lib` folder. This makes sense because there is no need for the system to scan through every single file when the probability for it to change is very small. People don't code directly in the `/lib` folder but only update when there is a new version of Ionic. However, there is some flexibility to change this.

You can specify a `watchPatterns` property in the `ionic.project` file located in your project root to watch (or not watch) for specific changes:

```
{  
  "name": "myApp",  
  "app_id": "",  
  "watchPatterns": [  
    "www/**/*",  
    "!www/css/**/*",  
    "your_folder_here/**/*"  
  ]  
}
```

While the web server is running, you can go back to the IDE and continue coding. For example, let's open the `playlists.html` file under `/www/templates` and change the first line to this:

```
<ion-view view-title="Updated Playlists">
```

Go back to the web browser where Ionic opened the new page; the app interface will change the title bar right away without requiring you to refresh the browser. This is a very nice feature when there is a lot of back and between code changes and allows checking on how it works or looks in the app instantly.

## Viewing the app using iOS Simulator

So far you have been testing the web-app portion of Ionic. In order to view the app in the simulator, follow the next steps.

### How to do it...

1. Add the specific platform using:

```
$ ionic platform add ios
```



Note that you need to do the "platform add" before building the app.

```
$ ionic build ios
```

2. The last step is to emulate the app:

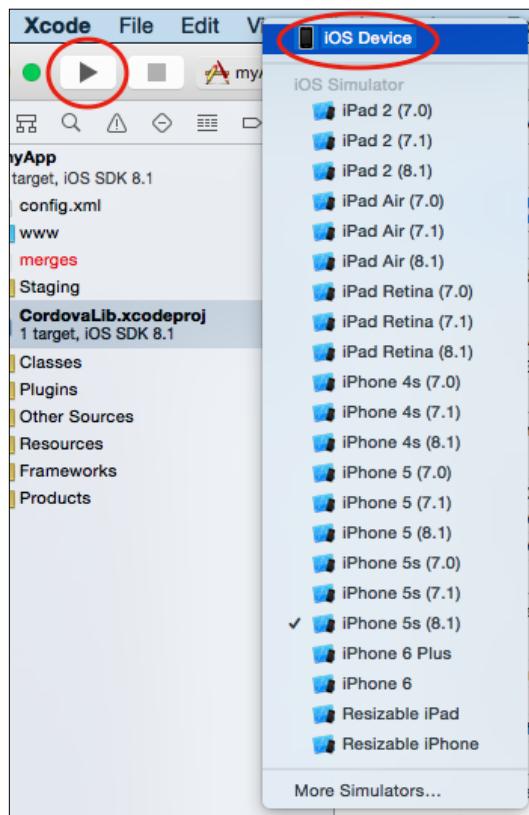
```
$ ionic emulate ios
```

## Viewing the app using Xcode for iOS

Depending on personal preference, you may find it more convenient to just deploy the app using `ionic ios --device` on a regular basis. This command line will push the app to your physical device connected via USB without ever running Xcode. However, you could run the app using Xcode (in Mac), too.

### How to do it...

1. Go to the `/platforms/ios` folder.
2. Look for the folder with `.xcodeproj` and open in Xcode.
3. Click on the iOS Device icon and select your choice of iOS Simulator.

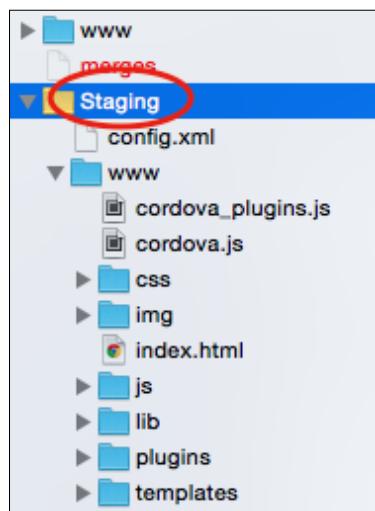


4. Click on the **Run** button and you should be able to see the app running in the simulator.

## There's more...

You can connect a physical device via a USB port and it will show up in the iOS Device list for you to pick. Then you can deploy the app directly on your device. Note that iOS Developer Membership is required for this. This method is more complex than just viewing the app via a web browser.

However, it's a must when you want to test your code related to device features such as camera or maps. If you change code in the `/www` folder and want to run it again in Xcode, you have to do `ionic build ios` first, because the running code is in the `Staging` folder of your Xcode project:



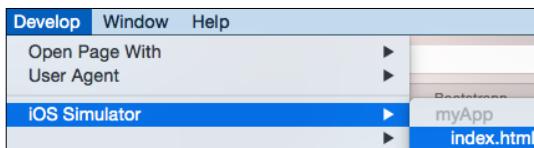
For debugging, the Xcode Console can output JavaScript logs as well. However, you could use the more advanced features of Safari's Web Inspector (which is similar to Google Chrome's Developer Tools) to debug your app. Note that only Safari can debug a web app running on a connected physical iOS device because Chrome does not support this on a Mac.

It's simple to enable this capability:

1. Allow remote debugging for an iOS device by going to **Settings | Safari | Advanced** and enable **Web Inspector**.



2. Connect the physical iOS device to your Mac via USB and run the app.
3. Open the Safari browser.
4. Select **Develop**, click on your device's name (or iOS Simulator), and click on `index.html`.



Note: If you don't see the **Develop** menu in Safari, you need to navigate to menu **Preferences | Advanced** and check on **Show Develop** menu in menu bar.

Safari will open a new console just for that specific device just as it's running within the computer's Safari.

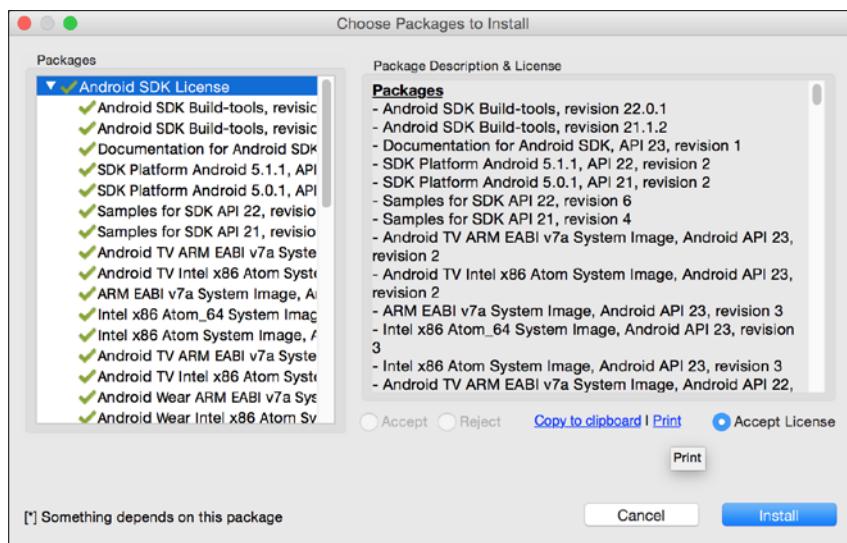
## Viewing the app using Genymotion for Android

Although it's possible to install the Google Android simulator, many developers have inconsistent experiences on a Mac computer. There are many commercial and free alternatives that offer more convenience and a wide range of device support. Genymotion provides some unique advantages such as allowing users to switch Android model and version, supporting networking from within the app, and allowing SD card simulation.

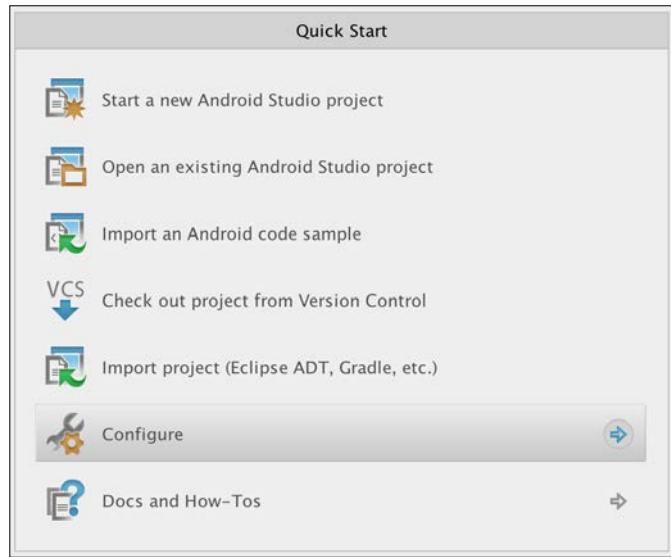
In this recipe, you will learn how to set up an Android developer environment (on a Mac in this case) first. Then you will install and configure Genymotion for mobile app development.

### How to do it...

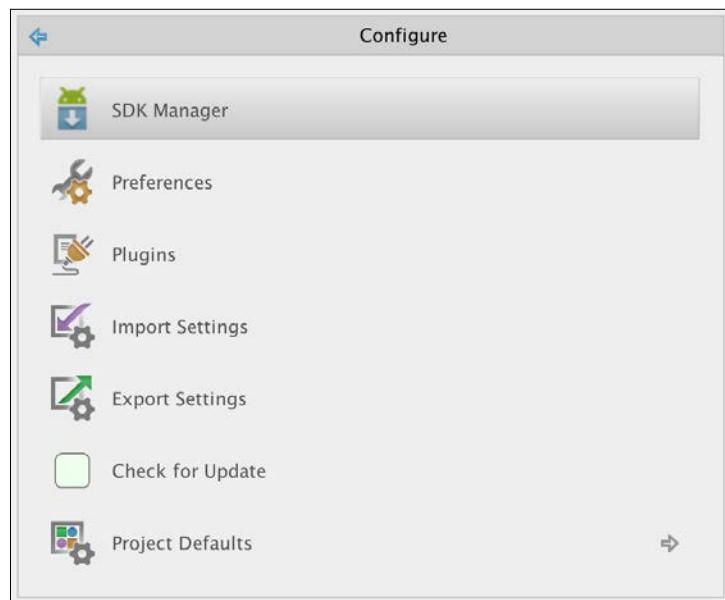
1. The first step is to set up the Android environment properly for development. Download and install Android Studio from <https://developer.android.com/sdk/index.html>.
2. Run Android Studio.
3. You need to install all required packages such as the Android SDK. Just click on Next twice at the Setup Wizard screen and select the **Finish** button to start packages installation.



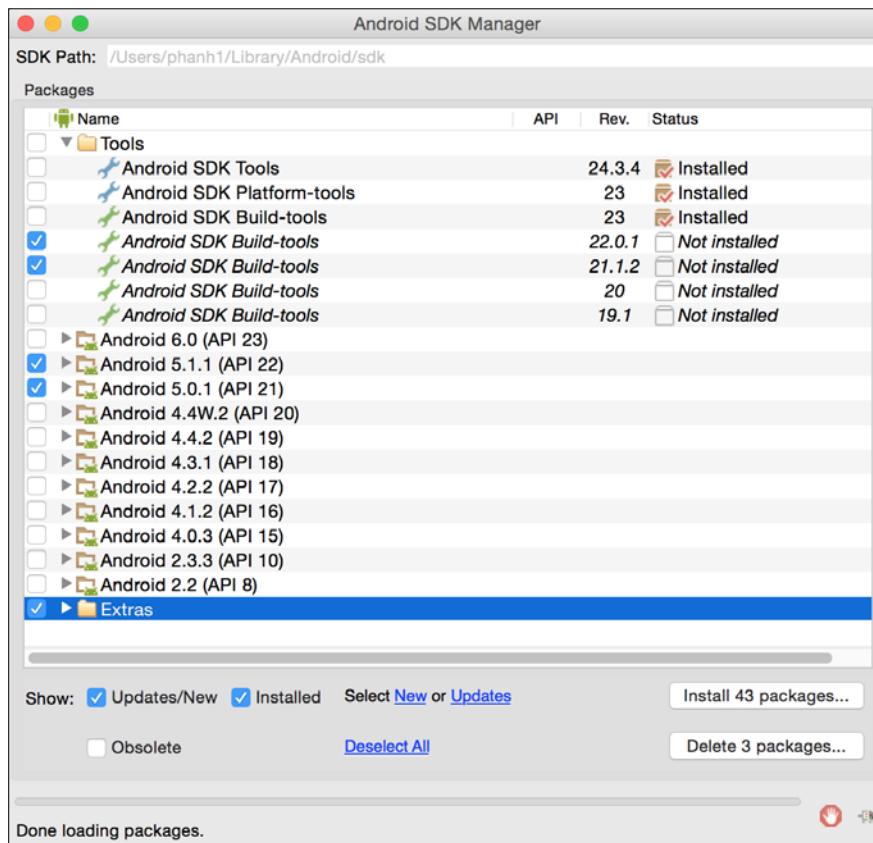
4. After installation is complete, you need to install additional packages and other SDK versions. At the **Quick Start** screen, select **Configure**:



5. Then select **SDK Manager**:

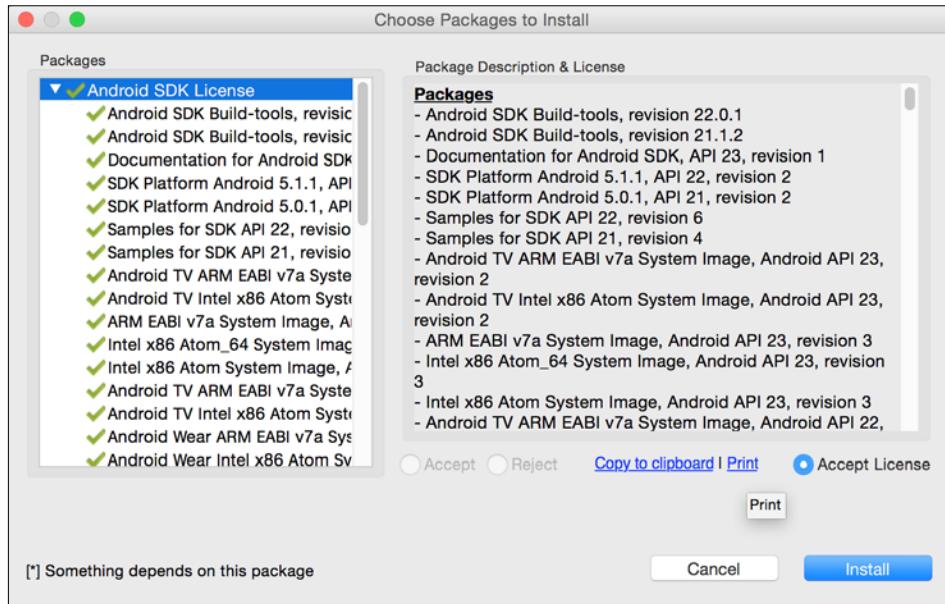


6. It's a good practice to install a previous version such as Android 5.0.1 and 5.1.1. You may also want to install all Tools and Extras for later use.



7. Select the **Install packages...** button.

8. Check the box on **Accept License** and click on **Install**.



9. The SDK Manager will give you SDK Path on the top. Make a copy of this path because you need to modify the environment path.

10. Go to Terminal and type:

```
$ touch ~/.bash_profile; open ~/.bash_profile
```

11. It will open a text editor to edit your bash profile file. Insert the following line where /YOUR\_PATH\_TO/android-sdk should be the SDK Path that you copied earlier:

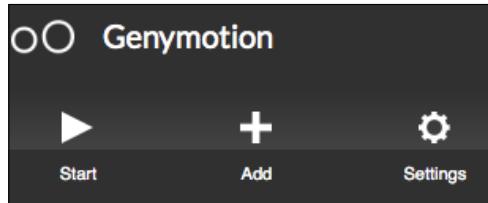
```
export ANDROID_HOME=/YOUR_PATH_TO/android-sdk
export PATH=$ANDROID_HOME/platform-tools:$PATH
export PATH=$ANDROID_HOME/tools:$PATH
```

12. Save and close that text editor.

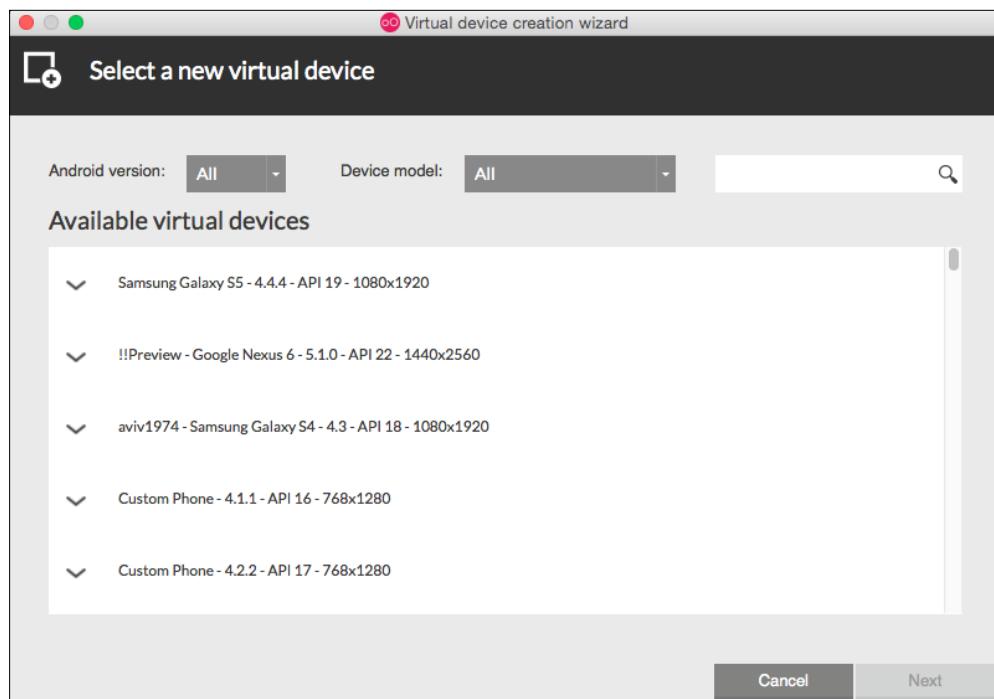
13. Go back to Terminal and type:

```
$ source ~/.bash_profile
$ echo $ANDROID_HOME
```

14. You should see the output as your SDK Path. This verifies that you have correctly configured the Android developer environment.
15. The second step is to install and configure Genymotion. Download and install Genymotion and Genymotion Shell from Genymotion.com.
16. Run Genymotion.
17. Select the **Add** button to start adding a new Android device.



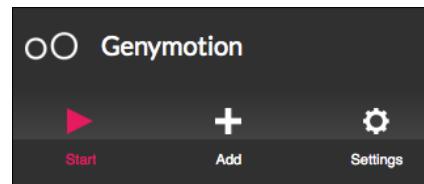
18. Select a device you want to simulate. In this case, let's select Samsung Galaxy S5:



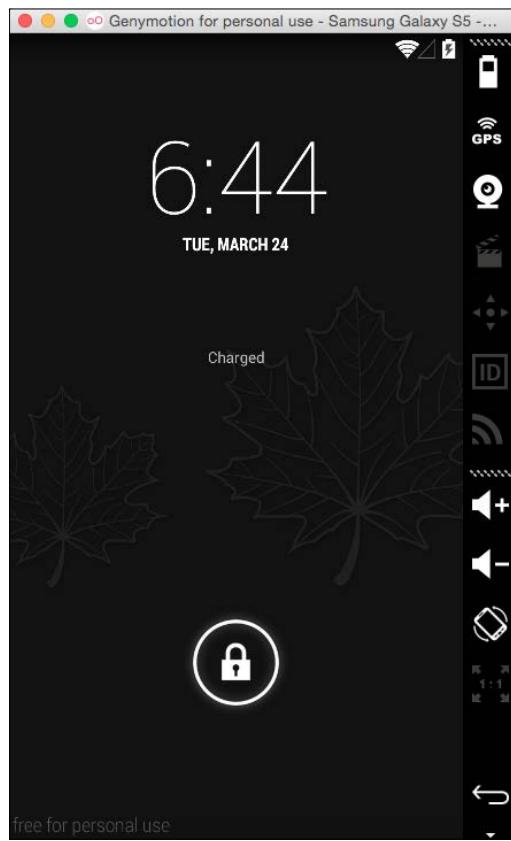
19. You will see the device being added to "Your virtual devices". Click on that device:



20. Then click on **Start**.



21. The simulator will take a few seconds to start and will show another window. This is just a blank simulator without your app running inside yet.

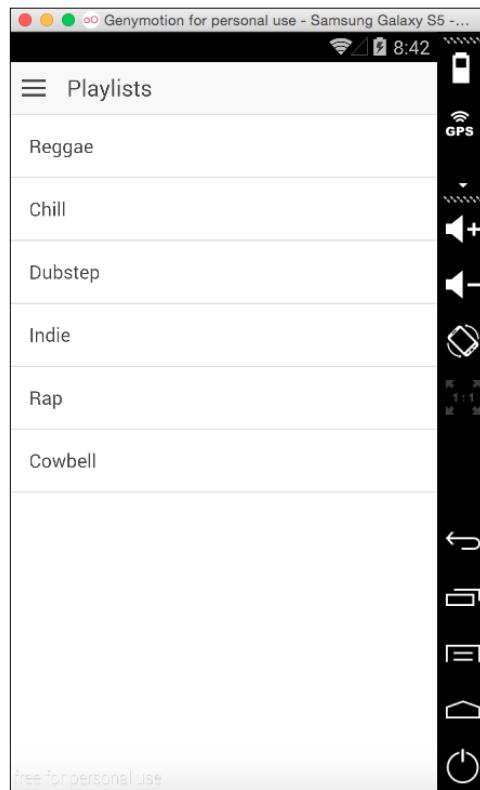


22. Run Genymotion Shell.
23. From Genymotion Shell, you need to get a device list and keep the IP address of the device attached, which is Samsung Galaxy S5. Type devices list:

```
Genymotion Shell > devices list
Available devices:

  Id | Select | Status | Type | IP Address | Name
+---+-----+-----+-----+-----+
  0 | *     | On   | virtual | 192.168.56.101 | Samsung Galaxy S5 -
4.4.4 - API 19 - 1080x1920
```

24. Type adb connect 192.168.56.101 (or whatever the IP address was you saw earlier from the devices list command line).
25. Type adb devices to confirm that it is connected.
26. Type ionic platform add android to add Android as a platform for your app.
27. Finally, type ionic run android.
28. You should be able to see the Genymotion window showing your app.



Although there are many steps to get this working, it's a lot less likely that you will have to go through the same process again. Once your environment is set up, all you need to do is to leave Genymotion running while writing code. If there is a need to test the app in different Android devices, it's simple just to add another virtual device in Genymotion and connect to it.

## Viewing the app using Ionic View

Ionic View is an app viewer that you can download from the App Store or Google Play. When you are in the development process and the app is not completed, you don't want to submit it to either Apple or Google right away but rather, limit access to your testers. Ionic View can help load your own app inside of Ionic View and make it behave like a real app with some access to native device features. Additionally, Ionic View lets you use your app on an iOS device without any certificate requirement.

Since Ionic View uses the Cordova inAppBrowser plugin to launch your app, all device features have to be "hacked" to make it work. Currently, Ionic View version 1.0.5 only supports SQLite, Battery, Camera, Device Motion, Device Orientation, Dialog/Notification, Geolocation, Globalization, Network Information, and Vibration. It's a good idea to check the updated support list before using Ionic View to ensure your app works properly.

### How to do it...

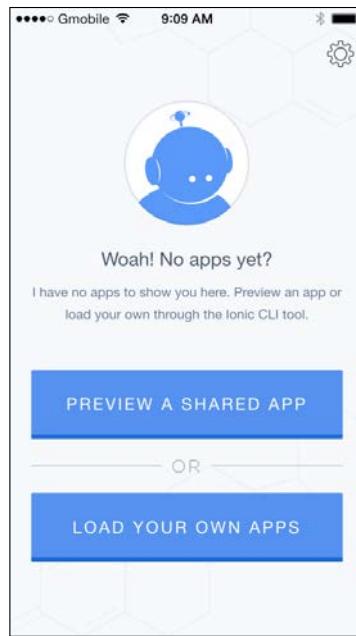
There are two ways to use Ionic View. You can either upload your own app or load someone else's App ID. If you test your own app, follow these steps:

1. Download Ionic View from either App Store or Google Play.
2. Make sure to register an account on ionic.io.
3. Go to your app's project folder.
4. Type `ionic upload`.
5. Enter your credentials.
6. The CLI will upload the entire app and give you the App ID, which is `152909f7` in this case. You may want to keep this App ID to share with other testers later.

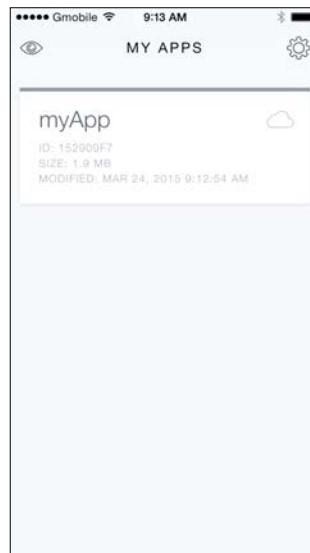
```
Uploading app...
Successfully uploaded (152909f7)

Share your beautiful app with someone:
$ ionic share EMAIL
```

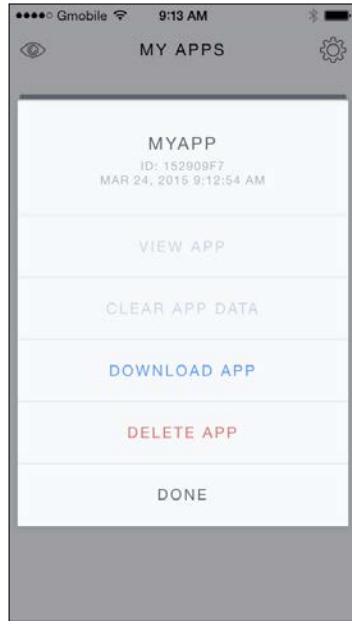
7. Open Ionic View and log in if you haven't done so.
8. Select **Load your own apps**.



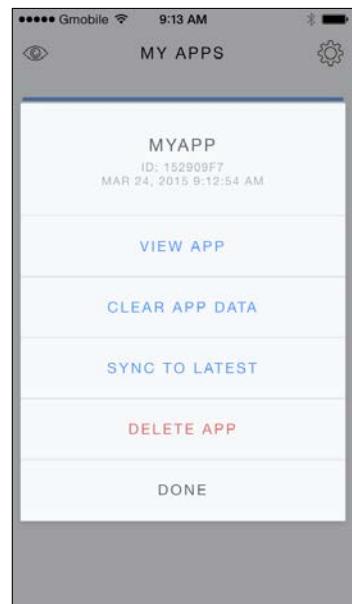
9. Now you should be able to see the app name in your **My Apps** page. Go ahead and select the app name (**myApp** in this case).



10. Select **Download App** to download the entire app in your Ionic View.



11. After the download process has completed, select **View App** to run the app.



12. You will see the app interface appears with initial instructions on how to exit the app. Since your app will cover the full screen of Ionic View, you need to swipe down by using three fingers to exit back to Ionic View.



If there is no code update, the process is the same except that you need to select **Sync to latest** at the menu.

In summary, there are several benefits of using Ionic View:

- ▶ It's convenient because there is only one command line to push the app.
- ▶ Anyone can access your app by entering the App ID.
- ▶ There is no need to even have iOS Developer Membership to start developing with Ionic. Apple has its own TestFlight app in which the use case is very similar.
- ▶ You can stay agile in the developer process by having testers test the app as you develop it.
- ▶ Ionic View has a wide range of device feature support and continues to grow.

## Customizing the app folder structure

The structure in starter templates may not be good enough depending on the app. It's important to understand its folder structure to allow further customization. Since the Ionic project is based on Cordova, most of what you see will be either iOS or Android related. This is the breakdown of what is inside the folder:

platforms/ (specific built code for iOS, Android, or Windows phone)	lib/
plugins/ (Cordova plugins)	ionic/ (CSS, fonts, JS, and SCSS from Ionic)
scss/	templates/ (UI-router templates)
ionic.app.scss (your app's custom Sass file)	index.html (main file)
www/	bower.json
css/ (your own css)	gulpfile.js
style.css (processed CSS file that will automatically be generated)	config.xml
img/ (your own images)	ionic.project
js/	package.json

### How to do it...

All application logic customization should be done in the /www folder as `index.html` is the bootstrap template. If you add in more JavaScript modules, you can put them in the /www/js/lib folder.

There is no need to modify the /platforms or /plugins folders manually unless troubleshooting needs to be done. Otherwise, the ionic or cordova CLI will automate the content inside those folders.



# 2

## Managing States and Navigation

In this chapter, we will cover the following tasks related to views and states:

- ▶ Creating a tab interface with nested views
- ▶ Creating a multistep form with validation

### Introduction

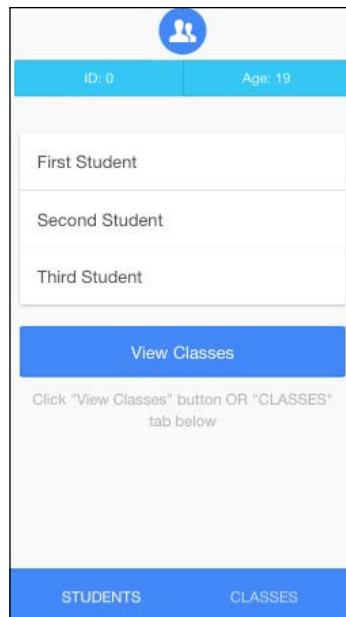
It's possible to write a simple app with a handful of pages. However, when the app grows, managing different views and their custom data at a specific time or triggered event could be very complex. Ionic comes with UI-Router by default. You should leverage this advanced routing management mechanism. In general, the following holds true:

- ▶ A view should have its own state, which is basically a JSON object
- ▶ A route (URL) will point to a view and its assigned controller
- ▶ A state and view should allow nested views so that you can manage hierarchy

Since Ionic introduces many new components, you have to understand how these components impact your app state hierarchy and when each state is triggered.

## Creating a tab interface with nested views

This recipe will explain how to work with the Ionic tab interface and expand it for other use cases. For example, it's possible to have additional views (that is, children) within each tab. Since each tab has its own view and state, you can also *watch* the tab change event. The following is the screenshot of the app:



In this app, you will learn the following:

- ▶ How to create nested views and states
- ▶ How to create a custom title per view
- ▶ How to watch for state changes and inject your own conditions depending on the new state

You will also use `$ionicLoading`, which is a very useful directive if you need to temporarily show some content on the screen (that is, a loading indicator).

### Getting ready

Since AngularJS UI-Router comes with the Ionic bundle, there is no need to download an external library. You can test this in a web browser as well.

## How to do it...

Here are the steps to create a tab interface with nested views:

1. Create a new app using the *blank* template and go into the folder:

```
$ ionic start Route blank  
$ cd Route
```

2. You need to set up the Sass dependencies in the following way because Ionic uses a number of external libraries for this:

```
$ ionic setup sass
```

3. Open the index.html file and replace the <body> tag with the following:

```
<ion-nav-bar class="bar bar-stable">  
  <ion-nav-back-button class="button-icon ion-arrow-left-c">  
  </ion-nav-back-button>  
</ion-nav-bar>  
  
<ion-nav-view></ion-nav-view>
```

This will set up the view for your navigation, which includes the top bar and content.

4. Open app.js and edit it with the following code:

```
var app = angular.module('starter', ['ionic'])  
  
app.config(function($stateProvider, $urlRouterProvider) {  
  $stateProvider  
    .state('app', {  
      url: "/app",  
      abstract: true,  
      templateUrl: "templates/app.html",  
      controller: "AppCtrl"  
    })  
    .state('app.students', {  
      url: "/students",  
      views: {  
        'students': {  
          templateUrl: "templates/students.html",  
          controller: 'StudentsCtrl'  
        }  
      }  
    })
```

```
.state('app.students.details', {
  url: "/details/:id/:age",
  views: {
    'details': {
      templateUrl: "templates/details.html",
      controller: 'StudentDetailsCtrl'
    }
  }
})
.state('app.classes', {
  url: "/classes",
  views: {
    'classes': {
      templateUrl: "templates/classes.html",
      controller: 'ClassesCtrl'
    }
  }
})
.state('app.classes.details', {
  url: "/details/:id"
});

$urlRouterProvider.otherwise("/app/students");
});
```

The `$stateProvider` object is used to set up the routing. You may realize that there are many template URLs that have not been defined yet. By default, the app will go to /app/students.

5. To define the templates, go back to `index.html` and insert the following code into it:

```
<script id="templates/app.html" type="text/ng-template">
<ion-tabs class="tabs-positive">

  <ion-tab title="STUDENTS" ui-sref="app.students">
    <ion-nav-view name="students"></ion-nav-view>
  </ion-tab>

  <ion-tab title="CLASSES" ui-sref="app.classes">
    <ion-nav-view name="classes"></ion-nav-view>
  </ion-tab>

</ion-tabs>
</script>

<script id="templates/students.html" type=
"text/ng-template">
```

```
<ion-view view-title="{{ title }}">
  <div class="bar bar-stable bar-subheader">
    <ui-view name="details"/>
  </div>
  <ion-content class="padding content-stable
has-subheader">
    <div class="card">
      <div class="item item-text-wrap"
ui-sref="app.students.details({id: 0, age: 19})">
        First Student
      </div>
      <div class="item item-text-wrap"
ui-sref="app.students.details({id: 1, age: 21})">
        Second Student
      </div>
      <div class="item item-text-wrap"
ui-sref="app.students.details({id: 2, age: 25})">
        Third Student
      </div>
    </div>
    <button class="button button-block
button-positive" ui-sref="app.classes">
      View Classes
    </button>
    <div class="hint">
      Click "View Classes" button OR "CLASSES" tab below
    </div>
  </ion-content>
</ion-view>
</script>

<script id="templates/details.html" type=
"text/ng-template">
  <div class="button-bar">
    <a class="button button-calm">ID: {{ id }}</a>
    <a class="button button-calm">Age: {{ age }}</a>
  </div>
</script>

<script id="templates/classes.html" type=
"text/ng-template">
  <ion-view view-title="{{ title }}">
    <ion-content class="padding content-stable">
      <div class="hint">
```

```
<b>NOTE:</b> No alert showing for
"app.classes.details" when clicking below
and no "ui-view" (optional)
</div>
<div class="card">
    <div class="item item-text-wrap"
        ui-sref-active-eq="item-active"
        ui-sref="app.classes.details({id:0})">
        Math
    </div>
    <div class="item item-text-wrap"
        ui-sref-active-eq="item-active" ui-sref=
        "app.classes.details({id:1})">
        English
    </div>
    <div class="item item-text-wrap"
        ui-sref-active-eq="item-active"
        ui-sref="app.classes.details({id:2})">
        Science
    </div>
</div>
<button class="button button-block
button-positive" ng-click="gotoStudents()">
    View Students
</button>
<div class="hint">
    Click "View Students" button OR "STUDENTS"
    tab below
</div>
</ion-content>
</ion-view>
</script>
```

The templates can be independent files, or they can be included in the `index.html` file itself as a `<script>` tag. You just need to reference it using the `id` attribute.

6. Add the following controller code in `app.js` for the **Students** and **Classes** tab:

```
app.controller('StudentsCtrl', function($scope) {
    $scope.title = '<div class="round-icon"><i class="icon
    ion-person-stalker"></i></div>';
});

app.controller('StudentDetailsCtrl', function($scope,
$stateParams) {
```

```
$scope.id = $stateParams.id;
$scope.age = $stateParams.age;
});

app.controller('ClassesCtrl', function($scope, $state) {
  $scope.title = '<div class="round-icon"><i class="icon ion-university"></i></div>';

  $scope.gotoStudents = function() {
    $state.go('app.students');
  }
});
```

Note that `StudentDetailsCtrl` is assigned to `templates/details.html` in the preceding code. So, it's as simple as passing the scope variables to `render` the data in the view.

7. One more thing that you need to do in `app.js` is create a high-level `AppCtrl` object to detect the state change. This is done by using `$rootScope.$on` for `$stateChangeSuccess`, as follows:

```
app.controller('AppCtrl', function($scope, $rootScope,
  $ionicLoading, $timeout) {

  $rootScope.$on('$stateChangeSuccess', function(event,
  toState, toParams, fromState, fromParams) {

    if (toState.name == 'app.classes.details')
      return;

    $ionicLoading.show({
      template: '<b>Previous state:</b> ' + fromState.name
      + '<br/><b>Current state</b>: ' + toState.name,
      noBackdrop: true
    });

    $timeout(function() {
      $ionicLoading.hide();
    }, 1000);
  });
});
```

8. To make the app look nice, you can customize it by using the following classes in the `ionic.app.scss` file under `/scss`:

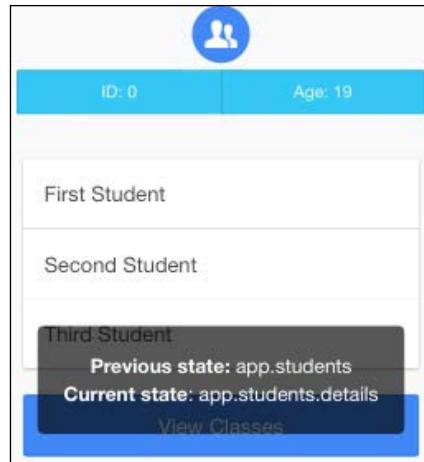
```
.bar .title {  
  overflow: visible!important;  
}  
  
.round-icon {  
  background-color: $positive;  
  border-radius: 50%;  
  width: $bar-height;  
  height: $bar-height;  
  margin-left: auto;  
  margin-right: auto;  
}  
  
.round-icon > i.icon {  
  color: white;  
  font-size: 28px;  
}  
  
.content-stable{  
  background-color: $stable  
}  
  
.hint {  
  color: #aaa;  
  font-size: 14px;  
  text-align: center;  
}  
  
.item-active {  
  background-color: darken($stable, 15%);  
}
```

One creative use of classes is shown by the use of the `round-icon` one, as it created a circle around the Ionic icon using `border-radius`.

9. Run the app by using the following command:

```
$ ionic serve
```

If you click on a student's name, it will switch to the `app.students.details` state and pass the parameters (`id` and `age`) to that state. You will also see the loading screen pop up to indicate the previous and current state.



## How it works...

At a high level, this is how the app is structured:

- ▶ The `<ion-nav-view></ion-nav-view>` tag will be replaced by the `app.html` template.
- ▶ The `app.html` template basically contains just a *skeleton* of `<ion-tabs>`. For each tab, the content will be replaced in `<ion-nav-view name="students"></ion-nav-view>` and `<ion-nav-view name="classes"></ion-nav-view>`.
- ▶ Each tab has to start with `<ion-view>` as the parent.
- ▶ For the **Students** tab, there are actually *detail* views nested inside. That's why you use `<ui-view name="details"/>` so that the additional children can replace it.
- ▶ When you click on a student's name, it actually triggers `ui-sref` directly and passes an expression such as `app.students.details({id: 0, age: 19})`. You basically treat each state name as a function and pass the parameters as a JSON. AngularJS UI-Router will take care of the rest.
- ▶ You must also tell `ion-tabs` about what to do after each tab in the bottom bar is clicked. That's why you must assign `ui-sref` for the `<ion-tab>` tag as well.

In Ionic, you cannot pass HTML code in the `view-title` directive. The workaround is to fill it with a `scope` variable in the following way so that it can be updated with HTML from the controller:

```
$scope.title = '<div class="round-icon"><i class="icon ion-person-stalker"></i></div>';
```

If the user clicks on the button at the bottom to go to another page, it is actually detected automatically by the Ionic tab directive to update the view.

Every time a state is changed, it will trigger a `$stateChangeSuccess` event after the change is completed. Watching for this event can be tricky because you have to set `$rootScope.$on` at the *topmost* level (that is, the parent controller). The reason behind this is that this binding must be persistent during navigation. Every time the app state is changed, there is no guarantee that the controller will stay in the memory. It can be destroyed, and it may lose all the binding. The parent controller will always remain persistent during the app's usage.

### See also

For further usage of AngularJS UI-Router, you can check out the GitHub repository at <https://github.com/angular-ui/ui-router>.

## Creating a multistep form with validation

Forms are everywhere on the web as well as in mobile apps. If you go through a registration process, it's done using a form. In a shopping cart solution, the user also steps through a multipage form that consists of address, payment information, confirmation page, and so on.

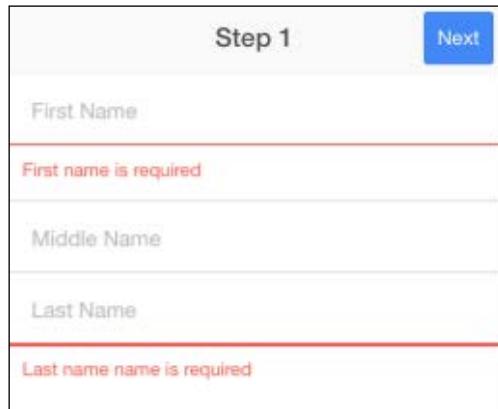
The example in this recipe will explain how to create a form for your app that can be placed in multiple pages and which can have a specific validation for each page. This may sound simple, but it can get complex when there are many pages involved and the user must be available to navigate back and forth. However, you give a lot of flexibility to the users, which results in a better experience.

The app will have four steps. Let's go through the app's functionality:

1. The first step will require three text fields with one field as optional:

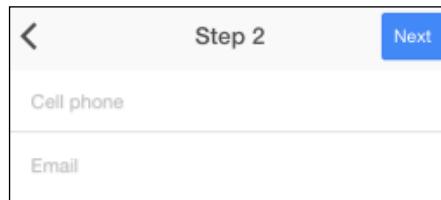
The diagram shows a wireframe of a multistep form. At the top, the title "Step 1" is centered above a blue rectangular button with the word "Next" in white. Below the title, there are three horizontal input fields. The first field is labeled "First Name", the second "Middle Name", and the third "Last Name". Each input field has a placeholder text inside it.

If the user does not fill in the **First Name** and **Last Name** fields, they will see the following error:



The screenshot shows a mobile-style form titled "Step 1". It has a "Next" button in the top right corner. There are three input fields: "First Name", "Middle Name", and "Last Name". The "First Name" field has a red border and the error message "First name is required". The "Last Name" field also has a red border and the error message "Last name name is required".

2. Clicking on **Next** will lead to **Step 2**:



The screenshot shows a mobile-style form titled "Step 2". It has a back arrow icon on the left and a "Next" button in the top right corner. There are two input fields: "Cell phone" and "Email".

There is no required field here, but once the user fills in their email ID, it must be validated:

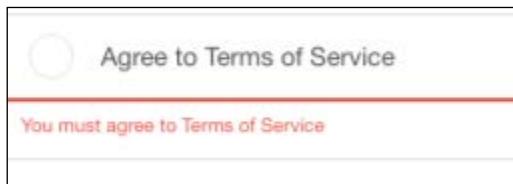


The screenshot shows a simple validation message box. It contains the word "test" in the top field and the error message "Email address is invalid" in the bottom field.

3. In **Step 3**, the user must check off a checkbox to agree to some terms of service:



Otherwise, it will show an error if the user tries to click on **Done**:



At any step, the user can go back and edit the previous page without an issue. This is usually very difficult if you are not using a Single Page Application framework such as Ionic.

## Getting ready

You don't need to test this on a physical device or even a backend server. This example will just simulate the saving of data in the memory and allow you to reset the app at the last step.

## How to do it...

Follow these steps:

1. Create a new app using a blank template and go into that folder, as follows:

```
$ ionic start MultistepForm blank  
$ cd MultistepForm
```

2. You need to set up the Sass dependencies in the following way:

```
$ ionic setup sass
```

3. Open the `index.html` file and replace the `<body>` tag with the following:

```
<body ng-app="starter" ng-controller="AppCtrl">
  <ion-nav-bar class="bar bar-stable">
    <ion-nav-back-button ng-show="!hideBackButton">
    </ion-nav-back-button>
  </ion-nav-bar>

  <ion-nav-view></ion-nav-view>
</body>
```

4. Let's ignore the JavaScript part for now and write the templates for the form. Since you need four steps, you have to create each step as a separate page. **Step 1** is structured as follows:

```
<script id="templates/step1.html" type="text/ng-template">
  <ion-view>
    <ion-nav-buttons side="primary">
      <h1 class="title">Step 1</h1>
      <button class="button button-positive"
        ng-click="submit()" ui-sref="step2">
        Next
      </button>
    </ion-nav-buttons>
    <ion-content>
      <form name="step1Form" ng-controller=
        "Step1FormCtrl" novalidate>
        <div class="list">
          <label class="item item-input">
            <input type="text" placeholder="First Name"
              name="firstname" ng-model="data.firstname"
              required>
          </label>
          <div class="item item-message"
            ng-if="step1Submitted &&
            step1Form.firstname.$error.required">
            First name is required
          </div>
          <label class="item item-input">
            <input type="text" placeholder="Middle Name"
              name="middlename" ng-model="data.middlename">
          </label>
          <label class="item item-input">
            <input type="text" placeholder="Last Name"
              name="lastname" ng-model="data.lastname"
              required>
          </label>
        </div>
      </form>
    </ion-content>
  </ion-view>
</script>
```

```
<div class="item item-message"
ng-if="step1Submitted &&
step1Form.lastname.$error.required">
    Last name is required
</div>
</div>
</form>
</ion-content>
</ion-view>
</script>
```

5. The code of **Step 2** is very much similar to that of **Step 1**, but it just has different fields:

```
<script id="templates/step2.html" type="text/ng-template">
<ion-view>
    <ion-nav-buttons side="primary">
        <h1 class="title">Step 2</h1>
        <button class="button button-positive"
ng-click="submit()" ui-sref="step3">
            Next
        </button>
    </ion-nav-buttons>
    <ion-content>
        <form name="step2Form" ng-controller=
"Step2FormCtrl" novalidate>
            <div class="list">
                <label class="item item-input">
                    <input type="tel" placeholder="Cell phone"
name="cell" ng-model="data.cell">
                </label>
                <label class="item item-input">
                    <input type="email" placeholder="Email"
name="email" ng-model="data.email">
                </label>
                <div class="item item-message"
ng-if="step2Submitted &&
step2Form.email.$error.email">
                    Email address is invalid
                </div>
            </div>
        </form>
    </ion-content>
</ion-view>
</script>
```

6. **Step 3** has the checkbox that needs to be checked off in order to proceed:

```
<script id="templates/step3.html" type="text/ng-template">
  <ion-view>
    <ion-nav-buttons side="primary">
      <h1 class="title">Step 3</h1>
      <button class="button button-positive"
        ng-click="submit()" ui-sref="done">
        Done
      </button>
    </ion-nav-buttons>
    <ion-content>
      <form name="step3Form" ng-controller="Step3FormCtrl"
        novalidate>
        <div class="list">
          <label class="item item-input">
            <textarea placeholder="Comments"
              name="comments" ng-model="data.comments"
              rows=10></textarea>
          </label>
          <div class="item item-checkbox">
            <label class="checkbox">
              <input type="checkbox" name="tos"
                ng-model="data.tos" ng-click="checkTos()">
            </label>
            Agree to Terms of Service
          </div>
          <div class="item item-message"
            ng-if="step3Submitted &&
            step3Form.tos.$error.agree">
            You must agree to Terms of Service
          </div>
        </div>
      </form>
    </ion-content>
  </ion-view>
</script>
```

7. Finally, **Step 4** is just about the **Done** page, where you render all the data in each page:

```
<script id="templates/done.html" type="text/ng-template">
  <ion-view>
    <ion-nav-buttons side="primary">
      <button class="button button-positive"
        ui-sref="step1" ng-click="reset()">
        Start Over
      </button>
    </ion-nav-buttons>
  </ion-view>
</script>
```

```
</button>
<h1 class="title">Thank You</h1>
</ion-nav-buttons>
<ion-content>
  <div class="list">
    <div class="item" ng-if="data.firstname">
      <b>First name:</b> {{ data.firstname }}
    </div>
    <div class="item" ng-if="data.middlename">
      <b>Middle name:</b> {{ data.middlename }}
    </div>
    <div class="item" ng-if="data.lastname">
      <b>Last name:</b> {{ data.lastname }}
    </div>
    <div class="item" ng-if="data.cell">
      <b>Cell:</b> {{ data.cell }}
    </div>
    <div class="item" ng-if="data.email">
      <b>Email:</b> {{ data.email }}
    </div>
    <div class="item" ng-if="data.comments">
      <b>Comments:</b> {{ data.comments }}
    </div>
    <div class="item" ng-if="data.tos">
      <b>Terms of Service:</b> {{ data.tos }}
    </div>
  </div>
</ion-content>
</ion-view>
</script>
```

8. Edit app.js and create the routes for all the steps, as follows:

```
var app = angular.module('starter', ['ionic'])

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

  $ionicConfigProvider.backButton.previousTitleText
  (false).text('');
  $ionicConfigProvider.views.swipeBackEnabled(false);
  $ionicConfigProvider.views.maxCache(0);

  $stateProvider
  .state('step1', {
```

```
url: "/step1",
  data: {
    step: 1
  },
  templateUrl: "templates/step1.html",
  controller: 'Step1Ctrl'
})
.state('step2', {
  url: "/step2",
  data: {
    step: 2
  },
  templateUrl: "templates/step2.html",
  controller: 'Step2Ctrl'
})
.state('step3', {
  url: "/step3",
  data: {
    step: 3
  },
  templateUrl: "templates/step3.html",
  controller: 'Step3Ctrl'
})
.state('done', {
  url: "/done",
  data: {
    step: 4
  },
  templateUrl: "templates/done.html",
  controller: 'DoneCtrl'
}) ;

$urlRouterProvider.otherwise("/step1");
});
```

9. Create `AppCtrl`, which is the topmost parent controller for the app, in the following way:

```
app.controller('AppCtrl', function($scope, $rootScope,
$ionicLoading, $timeout) {
  $scope.hideBackButton = false;
  $scope.data = {
    firstname: '',
    middlename: '',
    lastname: ''
```

```
    cell: '',
    email: '',
    comments: '',
    tos: false
};

$rootScope.$on('$stateChangeSuccess', function(event,
toState, toParams, fromState, fromParams) {
  if ((toState.name == 'done') || (toState.name ==
'step1'))
    $scope.hideBackButton = true;
  else
    $scope.hideBackButton = false;
});

});
```

This controller will handle the form object initialization. Note that the best practice should be to create a separate service/factory to keep the model of the form data. However, to make things simple in this example, you can leverage the AngularJS scope inheritance to share `$scope.data` in the entire app.

10. Each page will have two controllers: one to manage the entire page, and the other to handle only the form in that page. The reason you have to do this is that `<ion-content>` creates its own isolated scope. Since your **Next** button is in `<ion-buttons>`, which is outside `<ion-content>`, you cannot access child scopes from parent scopes.

```
app.controller('Step1Ctrl', function($scope) {
  $scope.step1Submitted = false;

  $scope.submit = function() {
    $scope.step1Submitted = true;
  }
});

app.controller('Step1FormCtrl', function($scope,
$rootScope, $state) {
  var validate = $rootScope.$on('$stateChangeStart',
  function(event, toState, toParams, fromState,
fromParams) {
    if (($scope.step1Form.$invalid) &&
      (toState.data.step > fromState.data.step))
      event.preventDefault();
  });

  $scope.$on('$destroy', validate);
});
```

11. A great thing about this method is that your controller tends to look very similar in each page. Here is the controller for **Step 2**:

```
app.controller('Step2Ctrl', function($scope) {
  $scope.step2Submitted = false;

  $scope.submit = function() {
    $scope.step2Submitted = true;
  }
});

app.controller('Step2FormCtrl', function($scope,
$rootScope, $state) {
  var validate = $rootScope.$on('$stateChangeStart',
  function(event, toState, toParams, fromState,
  fromParams) {
    if (($scope.step2Form.$invalid) && (toState.data.step
    > fromState.data.step))
      event.preventDefault();
  });

  $scope.$on('$destroy', validate);
});
```

12. The only difference in **Step 3** is related to how you handle the checkbox:

```
app.controller('Step3Ctrl', function($scope) {
  $scope.step3Submitted = false;

  $scope.submit = function() {
    $scope.step3Submitted = true;
  }
});

app.controller('Step3FormCtrl', function($scope,
$rootScope, $timeout) {
  $timeout(function() {
    $scope.step3Form.tos.$setValidity('agree',
    $scope.data.tos);
  });

  $scope.checkTos = function() {
    $scope.step3Form.tos.$setValidity('agree',
    $scope.data.tos);
  }

  var validate = $rootScope.$on('$stateChangeStart',
  function(event, toState, toParams, fromState,
  fromParams) {
```

```
        if ((($scope.step3Form.$invalid) &&
            (toState.data.step > fromState.data.step))
            event.preventDefault();
        });

        $scope.$on('$destroy', validate);
    });
}
```

13. The last **Done** page will basically allow the user to reset the data using angular.copy to avoid the *hijacking* of the object reference. Otherwise, you can lose the two-way binding with other pages.

```
app.controller('DoneCtrl', function($scope, $rootScope,
$ionicHistory) {
    $scope.reset = function() {
        angular.copy({
            firstname: '',
            middlename: '',
            lastname: '',
            cell: '',
            email: '',
            comments: '',
            tos: false
        }, $scope.data);
    }

    var validate = $rootScope.$on('$stateChangeSuccess',
        function(event, toState, toParams, fromState,
        fromParams) {
            $ionicHistory.clearHistory();
        });

    $scope.$on('$destroy', validate);
});
}
```

14. To handle the error message style, edit ionic.app.scss and insert the following code in it:

```
.item-message {
    background-color: $base-background-color;
    color: $assertive !important;
    font-size: 12px;
    padding: 5px 0px 10px 10px !important;
    border-top: 2px solid $assertive;
    white-space: normal;
}
```

15. From the command line, you can test the app using the following command:

```
$ ionic serve
```

#### Downloading the example code



You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## How it works...

To understand how a multistep form works, let's start with the routing system first. When you define a route, you actually assign a piece of data to each route as a step number. This is done using the `data` key, as follows:

```
$stateProvider
.state('step1', {
  url: "/step1",
  data: {
    step: 1
  },
  templateUrl: "templates/step1.html",
  controller: 'Step1Ctrl'
})
```

The reason you need to have this step number is to give the user the flexibility to go back to the previous page without having to complete the form in the current page. You will check whether the user goes back or forward using this step number.

To make a field required, you just need to put the `required` attribute in that `<input>` element, as follows:

```
<input type="text" placeholder="First Name" name="firstname"
ng-model="data.firstname" required>
```

However, in terms of validation, there are two things that you need to do:

```
<div class="item item-message" ng-if="step1Submitted &&
step1Form.firstname.$error.required">
  First name is required
</div>
```

First, you need to make sure that the user actually submitted the form by checking the Boolean value for `step1Submitted`. You may realize that there is no `submit` event here, since the app is just changing the state from one page to another. Each state change is basically a `submit` event (except if the user goes back to a previous page). This is why you cannot use the default `submit` event within AngularJS for validation.

The second thing that you should check for validation is any node in the `$error` object of the field. Keep in mind that `step1Form.firstname` here is the `name` attribute of the element, not the `ng-model` directive. Since there may be more validation errors, the only thing that you need to check here is the `required` error.

In the `email` input, the `type` attribute basically dictates its own validation for the email format, as follows:

```
<label class="item item-input">
  <input type="email" placeholder="Email" name="email"
    ng-model="data.email">
</label>
```

In this case, you check for `step2Form.email.$error.email` in the second step.

Handling the checkbox validation can get very tricky. What you can do is detect the click in that box and validate whether it's true or false, as follows:

```
<input type="checkbox" name="tos" ng-model="data.tos"
  ng-click="checkTos()">
```

The validation uses the `$setValidity` function from AngularJS to set the value, as follows:

```
$scope.step3Form.tos.$setValidity('agree', $scope.data.tos);
```

So, if this is invalid, the `step3Form.tos.$error.agree` variable will be `true`.

To handle the validation before the state change, you must watch the `$stateChangeStart` event:

```
var validate = $rootScope.$on('$stateChangeStart',
  function(event, toState, toParams, fromState, fromParams) {
    if (($scope.step1Form.$invalid) && (toState.data.step >
      fromState.data.step))
      event.preventDefault();
  });

```

If the form is invalid, you will trigger `preventDefault()` so that the user will not navigate to the next page.

Also, you should make sure that you destroy the watch at the `$rootScope` level when the controller is destroyed, as follows:

```
$scope.$on('$destroy', validate);
```

If you don't do this, you will end up having multiple *instances* of the function when the controllers are no longer around but the function is tight into `$rootScope` instead. In the `config()` function, this line purposely instructs Ionic not to cache any controller (to avoid issues with old data, binding, and so on):

```
$ionicConfigProvider.views.maxCache(0);
```

The last area to look at is the **Back** button. To avoid showing the **Back** button in the first and last screen (you don't want people to submit the entire form and go back to edit), you need to manually hide it when the correct state matches, as follows:

```
$rootScope.$on('$stateChangeSuccess', function(event, toState, toParams, fromState, fromParams) {
  if ((toState.name == 'done') || (toState.name == 'step1'))
    $scope.hideBackButton = true;
  else
    $scope.hideBackButton = false;
});
```

Then, from the view, you can use `ng-show` to manage the visibility, as follows:

```
<ion-nav-back-button ng-show="!hideBackButton">
```

## See also

It's highly recommended that you explore more on how AngularJS handles validation by visiting <https://docs.angularjs.org/guide/forms>.



# 3

## Adding Device Features Support

In this chapter, we will cover tasks related to leveraging Cordova plug-ins for native device functionalities:

- ▶ Taking a photo using the device camera
- ▶ Capturing video and allowing playback
- ▶ Composing an email with an attachment from an app
- ▶ Picking and adding a contact
- ▶ Adding Google Maps with geocoding support

### Introduction

In this chapter, you will learn how to access some common features of a device such as the camera, contact list, email, and map. Some of those features can be written in a JavaScript-only environment but the performance is not on a par with native support.

Cordova has a very well-supported community with many plugins. You may want to check out <http://plugins.cordova.io/> to understand what is out there. Luckily, you don't need to deal with those plugins directly. You can use the ngCordova (<http://ngcordova.com/docs/>) service on top of Cordova and AngularJS. Keep in mind that even if you use ngCordova, you still need the Cordova plugin because ngCordova just "Angular-izes" the way you interact with it.

## Taking a photo using the device camera

For this recipe, you will make an app to take a picture using the device camera or load an existing picture from the device album. The picture could be either in Base64 format or saved in a local filesystem of your app. Here is the high-level process:

- ▶ Access the Cordova Camera plugin to trigger camera capture and get the image back in the Base64 or file URI format
- ▶ Parse the Base64 data or URI on a `<img>` DOM object
- ▶ Display URI if it's in the URI format
- ▶ Capture an event of a toggle component
- ▶ Display long data (for example, URI) using horizontal scroll

### Getting ready

You should have a physical device ready in order to test camera capability. It's possible to just run it via an emulator, but the filesystem support might look different across platforms.

This section also requires Bower to be installed. Bower is a utility to help you manage packages for the frontend. It makes installing packages easier because you don't have to hunt all over the place for correct versioning and dependencies. If you don't have Bower yet, install it using the following command line:

```
$ npm install -g bower
```

### How to do it...

Here are the instructions to add camera support:

1. Start a blank project (for example, `Camera`) and go to that folder:

```
$ ionic start Camera blank  
$ cd Camera
```

2. Add the Cordova Camera plugin. The default template only has three plugins in the `/plugins` folder: `Console`, `Device`, and `Ionic.keyboard`.

```
$ cordova plugin add cordova-plugin-camera
```

3. Install ngCordova:

```
$ bower install ngCordova
```

You should be able to see a new folder, `org.apache.cordova.camera`, being added under the `/plugins` folder.

4. Open index.html and add ngCordova in the header:

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
```

Note that you have to put `ng-cordova.js` before `cordova.js`. Otherwise, you may have a compile error during the build process.

5. Add a controller in your main parent tag. You can just put it in the `<body>` tag as it's a simple app:

```
<body ng-app="starter" ng-controller="CameraCtrl">
```

6. This is basically the *skeleton* of the app: two buttons in a two-column grid, a list item below with two items (toggle switch and URI display), and `<img>` to show the photo from the device camera. Inside the `<body>` tag, put in the following code:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Camera</h1>
  </ion-header-bar>
  <ion-content>
    <div class="row">
      <div class="col">
        <button class="button button-calm button-full"
          ng-click="getPicture(1)">Show Camera</button>
      </div>
      <div class="col">
        <button class="button button-calm button-full"
          ng-click="getPicture(0)">Show Album</button>
      </div>
    </div>
    <ul class="list">
      <li>
        <ion-toggle ng-model="item.destinationFILE_URI"
          toggle-class="toggle-calm" ng-click=
          "clickToggle()">Return image file URI</ion-toggle>
      </li>
      <li>
        <div class="item item-icon-left" style=
        "background-color: #f8f8f8;">
          <i class="icon ion-link"></i>
          <ion-scroll direction="x">
            {{ item.imagePath }}
          </ion-scroll>
        </div>
      </li>
    </ul>
  </ion-content>
</ion-pane>
```

```

</ion-content>
</ion-pane>
```

Note that the Ionic grid system is very similar to Bootstrap where you have to declare rows and columns. However, Ionic makes it even simpler, so you don't need to specify the column ratio upfront. If you use the `col` class, it will automatically divide the width evenly.

7. Now let's write some code for the `CameraCtrl` controller. Open `app.js` and add your *skeleton* as shown next:

```
var app = angular.module('starter', ['ionic',
'ngCordova']);

app.controller('CameraCtrl', function($scope,
$cordovaCamera) {
});
```

Note that you must include `ngCordova` as a module dependency in order to use the `$cordovaCamera` service.

8. First, create an object to store the controller data:

```
$scope.item = {
  data: "",
  imagePath: "Photo capture as Base64",
  destinationFILE_URI: false
};
```



It's a good practice to store data in an object (that is, an item). Image data will be stored in `data`. Every time the user changes the destination type (base64 or file URI) or captures a new image, you need to update `item.imagePath` accordingly. The `destinationFILE_URI` object is just to save the current state of the toggle switch.

9. Now let's deal with the toggle switch by capturing the click or touch event:

```
$scope.clickToggle = function() {
  if ($scope.item.destinationFILE_URI)
    $scope.item.imagePath = "Photo capture as File URI";
  else
    $scope.item.imagePath = "Photo capture as Base64";
}
```

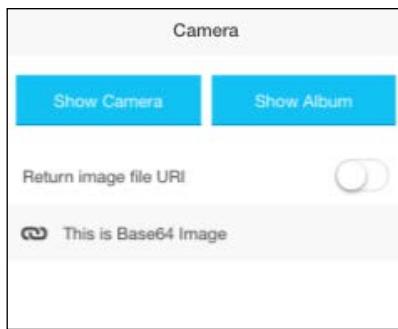
So basically every time the switch state is changed, the controller will update the `$scope.item.imagePath` variable.

10. The `getPicture()` function is the core part of this app because it does a lot of processing:

```
$scope.getPicture = function(sourceType) {
  var options = {
    quality : 50,
    allowEdit : true,
    correctOrientation: false,
    targetWidth: 640,
    targetHeight: 1080,
    destinationType: $scope.item.destinationFILE_URI ?
    Camera.DestinationType.FILE_URI :
    Camera.DestinationType.DATA_URL,
    sourceType : sourceType,
    encodingType: Camera.EncodingType.JPEG,
    saveToPhotoAlbum: false
  };

  $cordovaCamera.getPicture(options).then(function(imageData)
  {
    if ($scope.item.destinationFILE_URI) {
      $scope.item.data = imageData;
      $scope.item.imagePath = imagePath;
    } else {
      $scope.item.imagePath = "Photo capture as Base64";
      $scope.item.data = "data:image/jpeg;base64," +
      imageData;
    }
    console.log(imageData);
  }, function(err) {
    alert('Unable to take picture');
  });
}
```

11. When you run the app, you should see the app as shown in the following screenshot:



## How it works...

Let's start with step 6 of the preceding section.

Although this may look like a lot of stuffs, it's actually very simple. Let's start from the top down. The `<ion-header-bar>` tag is just a simple header bar to display the current page. You could even remove it. The `<ion-content>` tag is a wrapper for all app content. So this is the biggest area of your device screen.

```
<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Camera</h1>
  </ion-header-bar>
  <ion-content>
    <div class="row">
      <div class="col">
        <button class="button button-calm button-full"
          ng-click="getPicture(1)">Show Camera</button>
      </div>
      <div class="col">
        <button class="button button-calm button-full"
          ng-click="getPicture(0)">Show Album</button>
      </div>
    </div>
    <ul class="list">
      <li>
        <ion-toggle ng-model="item.destinationFILE_URI"
          toggle-class="toggle-calm" ng-click="clickToggle()">Return
          image file URI</ion-toggle>
      </li>
      <li>
        <div class="item item-icon-left"
          style="background-color: #f8f8f8;">
          <i class="icon ion-link"></i>
          <ion-scroll direction="x">
            {{ item.imagePath }}
          </ion-scroll>
        </div>
      </li>
    </ul>
    
  </ion-content>
</ion-pane>
```



For the image to appear in the `<img>` tag, you need to use `ng-src` instead of `src` alone. The reason is that the `<img>` attribute must be *controlled* by AngularJS in order to have live binding. That's it for the view layer.

The grid system in Ionic is super simple. You can just define it using the `.row` and `.col` classes. The concept is similar to the Bootstrap framework, but you don't have to specify the device size (that is, `-md`, `-lg`, and so on) or even the number of columns. There are two buttons and each is *bound* to the `getPicture()` function. The `<ion-toggle>` tag is simply just an AngularJS directive to create a nicer toggle switch for a mobile device. When the user clicks this toggle button, it will call the `clickToggle()` function.

`$cordovaCamera.getPicture()` is just an abstraction of `navigator.camera.getPicture()` from the Cordova Camera plugin. To post image data to the server, the common scenario is to upload the file from the filesystem. It's not a good idea to send data as Base64 because of the data size, which will increase the original binary size.

You just need to call `$cordovaCamera.getPicture()` with the `options` parameter and callbacks when there is success or an error. Note that it's a better option to reduce the image size and quality, otherwise the app will crash (due to the large memory requirements):

- ▶ `quality` is the quality of the saved image, expressed as a range of 0-100, where 100 is typically full resolution with no loss from file compression.
- ▶ If you want the photo to be automatically saved into the device's album, you can use `saveToPhotoAlbum: true` instead.
- ▶ `destinationType` could be `Camera.DestinationType.DATA_URL` (Base64) or `Camera.DestinationType.FILE_URI` (File URI).
- ▶ The callback will provide the `imageData` variable with the Base64 data or URI. Then you can assign it to the image's `ng-src` attribute.

You could get video in return but it only works if the video is already in the device (when `PictureSourceType` is `PHOTOLIBRARY` or `SAVEDPHOTOALBUM`). This tutorial does not go over the File plugin so that you can access existing photos, as it's a separate topic.

### There's more...

It is possible to create Instagram-like filter effects using just JavaScript. You can leverage an existing library such as Filterous (<https://github.com/girliemac/Filterous>) to modify the image canvas directly.

There is an Instagram plugin (<https://github.com/vstirbu/InstagramPlugin>) for Cordova on GitHub. You could write some extra code to pass the image to Instagram. The user must have Instagram installed in the phone first, though. This idea is nice when you plan to do some cool image processing (for example, adding funny text) before letting Instagram do the photo filter.

You can even use CSS3 filters (<http://codepen.io/SitePoint/full/KwmeJZ/>) to render in the frontend. However, the disadvantage of this method is that you cannot extract and save the binary of the new filtered image itself.

You could even add Cordova's Social Network plugin and post the resulting images to Twitter or Facebook.

## Capturing video and allowing playback

It's possible to capture video using Ionic. There is a different plugin called Cordova Media Capture, which allows more flexible access to audio and video capability. ngCordova also ported this plugin to create the `$cordovaCapture` service (<http://ngcordova.com/docs/plugins/capture/>).

This is very convenient because you just need to call any of those three functions:

- ▶ `$cordovaCapture.captureAudio`
- ▶ `$cordovaCapture.captureImage`
- ▶ `$cordovaCapture.captureVideo`

You could use this plugin for taking a photo as well, but it doesn't have a rich set of features like the Cordova Camera plugin does. The Media Capture plugin only provides access to the video camera to play back or update the resulting video file to a backend server. You cannot use this plugin to perform video processing such as adding text, animation, and special effects.

In this section, you will create a small app with a video capture button. You will learn how to create a video directive so that its playback content can be updated after recording a video.

### Getting ready

You need a physical device with functional video capability in order to go through this section.

### How to do it...

Here are the instructions:

1. Start a blank project (for example, Video-Capture) and go to that folder:

```
$ ionic start Video-Capture blank  
$ cd Video-Capture
```

2. Add the Cordova Media Capture plugin:

```
$ cordova plugin add org.apache.cordova.media-capture
```

3. Install ngCordova:

```
$ bower install ngCordova
```

4. Open index.html and add ngCordova in the header above cordova.js:

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
```

5. The body is very simple. You will create a VideoCtrl controller that triggers captureVideo() when the user clicks on a button. The <cordova-video> tag is a new directive to handle video playback.

```
<body ng-app="starter" ng-controller="VideoCtrl">
  <ion-pane>
    <ion-header-bar class="bar-stable">
      <h1 class="title">Video Capture</h1>
    </ion-header-bar>
    <ion-content>
      <button class="button button-calm button-full"
        ng-click="captureVideo()">Capture Video</button>
      <cordova-video src="data.videoPath"></cordova-video>
    </ion-content>
  </ion-pane>
</body>
```

6. Here is your VideoCtrl controller:

```
var app = angular.module('starter', ['ionic',
  'ngCordova']);

app.controller('VideoCtrl', function($scope,
  $cordovaCapture) {
  $scope.data = {
    videoPath: ""
  };

  $scope.captureVideo = $scope.captureVideo = function() {
    var options = { limit: 3, duration: 15 };

    $cordovaCapture.captureVideo(options).then(
      function(videoData) {
        // Success! Video data is here
        $scope.data.videoPath = "file://" +
        videoData[0].fullPath;
      }, function(err) {
        // An error occurred. Show a message to the user
        console.log(err);
      });
  }
});
```

Basically you just need to include ngCordova as a module dependency and declare that you plan to use \$cordovaCapture in the controller. Then once the button click is triggered, it will call \$cordovaCapture.captureVideo() to open up the device video recording. It's possible that the user will record multiple videos, so the returned object (for example, videoData) will always be an array. Your recorded video is stored locally in the app's folder, which you can access via videoData[0]. fullPath.

7. For the <cordova-video> directive, you just need to detect when a new video is recorded and add that to the playback:

```
app.directive("cordovaVideo", function () {
  return {
    restrict: 'AEC',
    scope: {src: '='},
    link: function(scope, element, attrs) {
      scope.$watch('src', function(newVal, oldVal) {
        if (scope.src != "") {
          // Create a div object
          var div = document.createElement('div');
          div.innerHTML = "<video id=\"myCordovaVideo\""
          controls>" +
            "<source src=\"" + scope.src +
            "\\" type=\"video/quicktime\">" +
            "</video>";

          // Delete previous video if exists
          var previousDiv =
            document.getElementById('myCordovaVideo');
          if (previousDiv)
            previousDiv.remove();

          // Append new <video> tag into the DOM
          element.append(div);
        }
      });
    }
  });
});
```

8. For minor styling effects of the video player, you can change `style.css` with the following code:

```
#myCordovaVideo {  
    background: red;  
    width: 100%;  
}
```

## How it works...

For step 7, there is a reason you have to write a separate directive to handle the `<video>` tag. AngularJS does not support live binding of the `<video>` tag, so if you write something such as `<video ng-src="yourObject" />`, it will not work. You also probably noticed that the directive uses `scope: { src: '=' }` as a way to direct the binding to the `data.videoPath` object. This is just a straight assignment by changing the `scope.src` reference to `data.videoPath` so that you can perform `$watch()` on this variable.

The main reason here is that the directive has its isolated scope, which is different from the controller. Another important area here is to call `remove()` on the previous DIV element inside this directive. The user may record one video after another. This is simply just a replacement of inner content to show the current video and delete previous ones.

Every time the user records a video, your `$cordovaCapture.captureVideo()` function will return an object in this format:

```
[ app.js:9  
  ▼ Object  
    end: 0  
    fullPath: "/private/var/mobile/Containers/Data/Application/2399FB21-EC24-46E3-BB97-F0A889738445/tmp/capture-T0x170078cc0.tmp.pC6clt/capturedvideo"  
    lastModified: null  
    lastModifiedDate: 1432526834000  
    localURL: "cdvfile:///localhost/temporary/capture-T0x170078cc0.tmp.pC6clt/capturedvideo.MOV"  
    name: "capturedvideo.MOV"  
    size: 268023  
    start: 0  
    type: "video/quicktime"  
    ▶ __proto__: Object ]
```

You can see that `fullPath` points to a folder with the unique ID of your app. This path is relative to your app. That's why it's necessary to prepend `file:/` in front of the string. Otherwise, the Cordova web app will not know where to find the video file.

Depending on the platform, some parameters such as limit or duration might not be supported by the Media Capture plugin. It's a good idea to visit the GitHub page for the latest information at <https://github.com/apache/cordova-plugin-media-capture>.

## There's more...

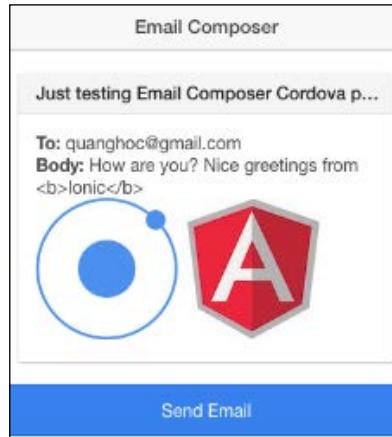
This section does not cover the process to send or update the video to a backend server because it's not unique to Cordova or Ionic. You just need to use the filesystem path pointing to the video file and upload it as normal.

There are many other free plugin and module alternatives to perform video playback. You could check out Videogular (<http://www.videogular.com/>), which is a very advanced video player written for AngularJS. There is also the Cordova Video Player plugin on GitHub if you want flexibility to access the native player (<https://github.com/macdonst/VideoPlayer>).

## Composing an email with an attachment from an app

Your use case may include the ability to let users compose their own email with populated fields. For example, after users perform some specific actions, the app automatically fills in the email receiver, subject, and body to have users send it off. This functionality makes it easy for users so that they don't have to manually compose the email. Many apps use this technique for the *tell a friend* feature.

The code example in this section will use Cordova's Email Composer plugin so that your Ionic app has a standard interface to communicate with the native device feature. You will understand how to attach files such as images to user emails. The image could be a local file in the app filesystem or Base64 string, which is a result from the device camera. The body could even contain HTML tags. Here is the screenshot of the app you are going to build:



However, keep in mind that there is no way to detect that the e-mail has been sent successfully. There are many real-world scenarios that you cannot control such as Internet connectivity (that is, Airplane mode), mail server error, and so on. The plugin only provides the callback function when the user closes down their email composer app.

## Getting ready

You will need a physical device to test an email. The device must have the email configured in order to send the email.

## How to do it...

Here are the steps for composing an email with an attachment from an app:

1. Create a blank Ionic app (for example, Email) and change the directory to that folder:

```
$ ionic start Email blank
```

2. Install the Email Composer plugin:

```
$ cordova plugin add https://github.com/katzer/cordova-plugin-email-composer.git
```

3. Install ngCordova:

```
$ bower install ngCordova
```

4. Open index.html and add ngCordova in the header:

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
```

5. You will need to add the EmailCtrl controller:

```
<body ng-app="starter" ng-controller="EmailCtrl">
```

6. Let's show the content of the email object in the view. Add the card layout as a way to organize this information:

```
<div class="card">
  <div class="item item-divider">
    {{ email.subject }}
  </div>
  <div class="item item-text-wrap">
    <strong>To:</strong> {{ email.to }}<br>
    <strong>Body:</strong> {{ email.body }}<br>
```

```


</div>
</div>
```

The email's to, subject, and body fields are very straightforward. However, this example will show you how to attach a local file (`ionic.png`) as well as a Base64 image. Since the Base64 string is a `Base64Icon` variable, you need to use `ng-src` to properly parse this variable. The `src` attribute can take either URL or Base64 data.

7. Add a button to trigger the email composer, which you will write a function for later:

```
<button class="button button-full button-positive"
ng-click="send()">Send Email</button>
```

8. After the user closes down the composer (regardless of whether the email is sent or not), show a "thank you" note by detecting the `thankYou` Boolean:

```
<p ng-if="thankYou" class="button button-clear
button-positive">
    Thank you!
</p>
```

9. Open `app.js` for editing. Make sure `ngCordova` is included as a dependency. You can remove the entire `run()` function as there is no need for it.

```
var app = angular.module('starter', ['ionic',
'ngCordova']);
```

10. Create the controller:

```
app.controller('EmailCtrl', function($scope,
$ionicPlatform, $cordovaEmailComposer, Base64Icon) {
});
```

In this controller, you need to do three things:

- ❑ Check to make sure emails are available on the device
- ❑ Initialize the email object with some data
- ❑ Create a `send()` function to trigger the email app

You will create a separate factory called `Base64Icon` later to store the Base64 string.

11. To check for email capability, you need to call the `isAvailable()` function. It's possible that the device is not set up for emails or the email app is not available for some reason.

```
$ionicPlatform.ready(function() {
  $cordovaEmailComposer.isAvailable().then(function() {
    // is available
    console.log('Email is available');
  }, function () {
    // not available
    alert('Email is not available');
  });
});
```

It's important to wrap `isAvailable()` around the `ready()` function. The reason is that when your app starts, it may take a bit of time to load all plugins.

12. Initialize two scope variables: `Base64Icon` and `email`:

```
$scope.Base64Icon = Base64Icon;
$scope.email = {
  to: 'youremail@gmail.com',
  // You can add cc or bcc field with array of emails
  // cc: 'someone@gmail.com',
  // bcc: ['test1@gmail.com', 'test2@gmail.com'],
  attachments: [
    'file:///img/ionic.png',
    // file:// is basically your www/ folder
    // You can include any file such as PDF
    // 'file:///README.pdf'
    "base64:icon.jpg://" + Base64Icon
    // Note that you must include file name for base64
    // such as icon.jpg
    // 'base64:icon.png//iVBORw0KGgoAAAANSUhEUg...',
  ],
  subject: 'Just testing Email Composer Cordova plugin',
  body: 'How are you? Nice greetings from <b>Ionic</b>',
  isHtml: true
};
```



You basically just assign the value of the Base64Icon factory into a scope variable so you can access it from the view. Attachment is probably the most difficult field to get right. You could attach a local file from the app folder. By default, everything is under the www folder. So to reference to the img folder, you just need to add file://img/ as the path.

For the Base64 image, you need to have "base64:icon.jpg://" as the prefix. The email composer requires a filename in order to attach. If you recall the view, the src attribute requires "data:image/jpg;base64," (don't forget the comma at the end) as the prefix instead. So that is the main difference.

13. Create the send() function:

```
$scope.send = function() {  
    $cordovaEmailComposer.open($scope.email).then(null,  
    function () {  
        // Callback when user cancelled or sent email  
        $scope.thankYou = true;  
    })  
}
```

This function calls \$cordovaEmailComposer.open() by passing the email object. Once the composer is closed, it assigns thankYou = true in order to show the note in the view.

14. You could include any Base64 string; the example in this book shows the Angular logo. Create services.js and make sure to include it in the index.html header:

```
app.factory('Base64Icon', function() {  
    return "/9j/4AAQSkZJRg..."  
});
```

15. You cannot run this app in your browser. So the alternative is to deploy the app via Xcode or TestFlight.

## How it works...

The \$cordovaEmailComposer factory only translates three functions from the plugin:

- ▶ isAvailable
- ▶ open
- ▶ addAlias

Keep in mind that although you can include HTML tags in the email body, it does not allow any CSS. Here is the list of tags that you can use:

• <code>&lt;a href="..."&gt;</code>	• <code>&lt;h4&gt;</code>
• <code>&lt;b&gt;</code>	• <code>&lt;h5&gt;</code>
• <code>&lt;big&gt;</code>	• <code>&lt;h6&gt;</code>
• <code>&lt;blockquote&gt;</code>	• <code>&lt;i&gt;</code>
• <code>&lt;br&gt;</code>	• <code>&lt;img src="..."&gt;</code>
• <code>&lt;cite&gt;</code>	• <code>&lt;p&gt;</code>
• <code>&lt;dfn&gt;</code>	• <code>&lt;small&gt;</code>
• <code>&lt;div align="..."&gt;</code>	• <code>&lt;strike&gt;</code>
• <code>&lt;em&gt;</code>	• <code>&lt;strong&gt;</code>
• <code>&lt;font size="..." color="..." face="..."&gt;</code>	• <code>&lt;sub&gt;</code>
• <code>&lt;h1&gt;</code>	• <code>&lt;sup&gt;</code>
• <code>&lt;h2&gt;</code>	• <code>&lt;tt&gt;</code>
• <code>&lt;h3&gt;</code>	• <code>&lt;u&gt;</code>

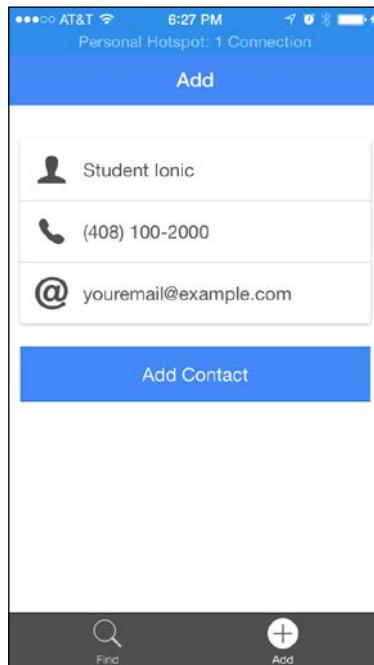
## See also

- ▶ You can read the code of this factory directly to understand it, at the following URL:  
<https://github.com/driftyco/ng-cordova/blob/master/src/plugins/emailComposer.js>
- ▶ To see the latest update of \$cordovaEmailComposer, you can visit the documentation page:  
<http://ngcordova.com/docs/plugins/emailComposer/>
- ▶ There are many other topics that have not been discussed in this section, such as specific problems per platform. The Email Composer's GitHub page is the best place to get the details:  
<https://github.com/katzer/cordova-plugin-email-composer>

## Picking and adding a contact

Your app sometimes may need to access the device contact list for searching, picking, or adding a contact. This ability can save users time instead of requesting them to enter the information manually.

In this section, you will learn how \$cordovaContacts works. Also the app will demonstrate the Ionic tabs feature by creating a navigation view container that has two tabs (a **Find** tab for picking a contact and an **Add** tab for saving a contact). Then UI-Router will wire each tab to a specific template. The following is the screenshot of the app:



## How to do it...

Here are the steps to follow this recipe:

1. Create a blank Ionic app (for example, Contacts) and change the directory to that folder:

```
$ ionic start Contacts blank
```

2. Install the Contacts plugin:

```
$ cordova plugin add org.apache.cordova.contacts
```

3. Install ngCordova:

```
$ bower install ngCordova
```

4. Open index.html and add ngCordova in the header:

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
```

5. You will need to add the ContactCtrl controller:

```
<body ng-app="starter" ng-controller="ContactCtrl">
```

6. Before writing the controller's code, you need to think how to structure the view. This app will have two tabs. Each tab has its own title bar and content. So let's create the *outer* frame using `<ion-nav-bar>` first:

```
<ion-nav-bar class="bar-positive">  
</ion-nav-bar>
```

```
<ion-nav-view></ion-nav-view>
```

The `<ion-nav-view>` tag is just a *blank* container like `ng-view`. You will use UI-Router to assign a template into this placeholder.

7. To use Ionic tabs, you need to put the structure in a separate template so that it can be injected into `<ion-nav-view>` at runtime. You could have it in a separate HTML file, but another option is to put the template inside the `<script>` tag:

```
<script id="templates/tabs.html" type="text/ng-template">  
  <ion-tabs class="tabs-dark tabs-icon-top">  
    <ion-tab title="Find" icon-on="ion-ios-search-strong"  
             icon-off="ion-ios-search" ui-sref="tabs.find">  
      <ion-nav-view name="find-tab"></ion-nav-view>  
    </ion-tab>  
  
    <ion-tab title="Add" icon-on="ion-ios-plus"  
             icon-off="ion-ios-plus-outline" ui-sref="tabs.add">  
      <ion-nav-view name="add-tab"></ion-nav-view>  
    </ion-tab>  
  </ion-tabs>  
</script>
```

To create tabs, you need to start with the grouping `<ion-tabs>` and then the individual tab item in `<ion-tab>`. The `title` attribute is for the *label* in each tab icon. The `ui-sref` attribute is a part of UI-Router that assigns a state name to that tab. This helps you avoid hardcoding the URL. You also need to give each child, `<ion-nav-view>`, a name such as `find-tab` or `add-tab`. UI-Router will use these names later in `app.js` to properly assign a template to each view.

8. Once you have the *structure*, let's work on the individual content of each tab:

```
<script id="templates/find.html" type="text/ng-template">  
  <ion-view title="Find">  
    <ion-content class="padding has-header">  
      <div class="list card">  
  
        <div class="item item-icon-left">
```

```
<i class="icon ion-person"></i>
{{ contactFind.name.givenName }} {{ contactFind.name.familyName }}
</div>

<div class="item item-icon-left">
  <i class="icon ion-ios-telephone"></i>
  {{ contactFind.phoneNumbers[0].value }}
</div>

<div class="item item-icon-left">
  <i class="icon ion-at"></i>
  {{ (contactFind.emails[0].value) || "Not Available" }}
</div>

</div>
<button class="button button-full button-positive"
ng-click="pickContact()">Pick Contact</button>
</ion-content>
</ion-view>
</script>
```

The first tab will allow users to pick a contact and render the information such as name and phone number. Each field can be displayed inside the Ionic's list card. Below the card is just a button to trigger the `pickContact()` function, where the Contacts app will appear for users to pick any contact.

You may notice that each tab's content starts with `<ion-view>` and then `<ion-content>` as a child. The `<ion-view>` wrapper is necessary for Ionic to assign a proper title text on the top bar. `<ion-content>` gives you flexibility to add padding on the top and/or bottom so that there is no content hidden from the header bar.

9. The second tab content will trigger the `addContact()` function to add a contact into the device's contact list.

```
<script id="templates/add.html" type="text/ng-template">
<ion-view title="Add">
  <ion-content class="padding has-header">
    <div class="list card">

      <div class="item item-icon-left">
        <i class="icon ion-person"></i>
        {{ contactSave.name.givenName }} {{ contactSave.name.familyName }}
```

```
</div>

<div class="item item-icon-left">
  <i class="icon ion-ios-telephone"></i>
  {{ contactSave.phoneNumbers[0].value }}
</div>

<div class="item item-icon-left">
  <i class="icon ion-at"></i>
  {{ contactSave.emails[0].value }}
</div>

</div>
<button class="button button-full button-positive"
ng-click="addContact()">Add Contact</button>
</ion-content>
</ion-view>
</script>
```

10. Now you need to edit `app.js` to add the controller. Let's start with the `app` module:

```
var app = angular.module('starter', ['ionic',
'ngCordova']);

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();
    }
  });
});
```

11. Then define the routing config:

```
app.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    .state('tabs', {
      url: "/tab",
```

```
        abstract: true,
        templateUrl: "templates/tabs.html"
    })
.state('tabs.find', {
    url: "/find",
    views: {
        'find-tab': {
            templateUrl: "templates/find.html"
        }
    }
})
.state('tabs.add', {
    url: "/add",
    views: {
        'add-tab': {
            templateUrl: "templates/add.html"
        }
    }
});
$urlRouterProvider.otherwise("/tab/find");
});
```

The state `tabs` is just a parent state to contain the `tabs` structure. Users cannot just arrive at this state without going to the child state. That's why one of the parameters has `abstract: true` value. Both `tabs.find` and `tabs.add` are children of `tabs`.

When you put a dot notation in the state name, UI-Router automatically interprets it as a child. Note that `find-tab` and `add-tab` are the names you declared in the `tabs.html` template earlier. The `otherwise()` function will tell the app to go to the `find` tab by default:

```
$urlRouterProvider.otherwise("/tab/find");
```

12. Create the controller and include `$cordovaContacts` as a dependency:

```
app.controller('ContactCtrl', function($scope,
$cordovaContacts) {
});
```

13. Within the controller, let's initialize your contact objects:

```
$scope.contactFind = {
    "name": {
        "givenName": "Not",
```

```
        "familyName": "Available"
    },
    "phoneNumbers": [
        {
            "value": "Not Available",
            "type": ""
        }
    ],
    "emails": [
        {
            "value": "Not Available"
        }
    ]
};

$scope.contactSave = {
    "name": {
        "givenName": "Student",
        "familyName": "Ionic"
    },
    "phoneNumbers": [
        {
            "value": "(408) 100-2000",
            "type": "mobile"
        }
    ],
    "emails": [
        {
            "value": "youremail@example.com"
        }
    ]
};
```

The `contactFind` object is used in the `Find` tab, whereas the `contactSave` object is used in the `Add` tab. This is just a basic example to demonstrate the ability to get and add a contact.

14. Write your `pickContact()` function:

```
$scope.pickContact = function() {
    $cordovaContacts.pickContact().then(function(result) {
        // Contact picked success
        console.log(result);
    })
};
```

```
$scope.contactFind = result;
}, function(err) {
// Contact picked error
alert('There is an error picking contact.
Please see console.log');
console.log(err);
});
};
```

This function simply triggers `$cordovaContacts.pickContact()` and assigns the result back into your scope object `$scope.contactFind`.

15. To add a contact, call `$cordovaContacts.save()` and pass the contact object:

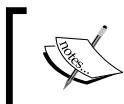
```
$scope.addContact = function() {
$cordovaContacts.save($scope.contactSave).
then(function(result) {
// Contact saved success
alert('The contact information has been saved');
console.log(result);
}, function(err) {
// Contact saved error
alert('There is an error saving contact.
Please see console.log');
console.log(err);
});
};
```

This step requires the user to authorize the app's access to the device's contact list. There is a possibility that the function may fail, so you need to catch that in an error callback. If for some reason you cannot see the new contact `Student Ionic` being added, it's probably because your device contact hides it. You just need to choose to show all contacts from all groups within the device's contact list app.

16. You cannot run this app in the browser as there won't be a contact list available. It's best if you load the app directly in the device or through TestFlight.

## How it works...

There are many fields that can be returned from a contact object. However, they are not supported consistently across all platforms. You may want to check the plugin developer page on GitHub for details at <https://github.com/apache/cordova-plugin-contacts/>.



There is no way to query all contacts and return an array. This is due to privacy. You can only ask the user to pick one contact at a time via the `pickContact()` function.

The user's mobile device may automatically convert a phone number string into a live *click-to-call* link. If this happens, the browser actually modified the DOM and your Angular binding won't work any more as it lost the reference to that new DOM. To prevent<sup>26</sup>

this scenario from happening, make sure to add the following `meta` tag:

```
<meta name="format-detection" content="telephone=no">
```

You could always manually add the link by using this syntax:

```
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

It's better to control this process to avoid your DOM being modified by the device.

## See also

To stay up to date with the current development of `$cordovaContacts`, you can visit this:

<http://ngcordova.com/docs/plugins/contacts/>

## Adding Google Maps with geocoding support

Many mobile apps today utilize different mapping features such as showing the current location, creating routes, and providing a suggestive business search. This recipe will show you how to use the Cordova Google Maps plugin to provide mapping support. For more information, visit <https://github.com/wf9a5m75/phonegap-googlemaps-plugin>.

You will create an app that can:

- ▶ Display Google Maps in full screen with the ability to detect the current device location
- ▶ Perform geocoding to find the address of any coordinate
- ▶ Add a marker with any text
- ▶ Abstract all mapping functions into a new `<ion-map>` directive and `$ionicMap` delegate

It is possible to use the HTML5 and JavaScript version of geolocation and maps, instead of Cordova plugins. However, you will see a negative impact on performance. It's very obvious that if you use the SDK, map rendering and optimization tends to be faster. In addition, HTML5 geolocation sometimes has some strange bugs that require the user to accept permission twice: once for the app and once for the inside browser object.

## Getting ready

The Google Maps plugin requires a Google Maps API key for your project. You need a Google account and login credentials to get started.

1. Navigate to the Google APIs Console at <https://code.google.com/apis/console/>.
2. Create a project if you don't have one yet. Just fill in the fields required:

New Project

Project name ?

Project ID ?

Show advanced options...

3. You need to enable the Google Maps SDK for iOS or the Google Maps Android API, or both. It depends on how many platforms you plan to support. Let's select the Google Maps SDK for iOS for this example:

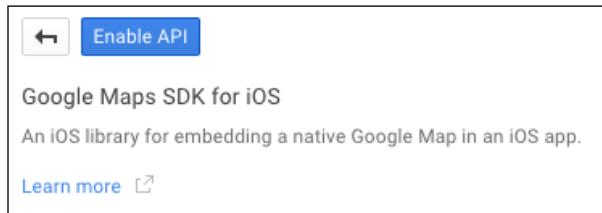
API Library Enabled APIs (6)

Search all 100+ APIs

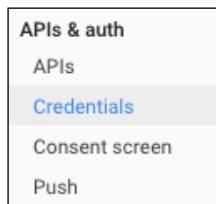
Popular APIs

 Google Cloud APIs	 Google Maps APIs
Compute Engine API	Google Maps Android API
BigQuery API	Google Maps SDK for iOS
Cloud Storage API	Google Maps JavaScript API
Cloud Datastore API	Google Maps Embed API
Cloud Deployment Manager API	Google Places API for Android
Cloud DNS API	Geocoding API
More	More

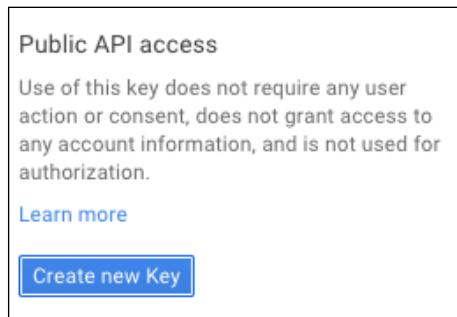
4. Click on the **Enable API** button:



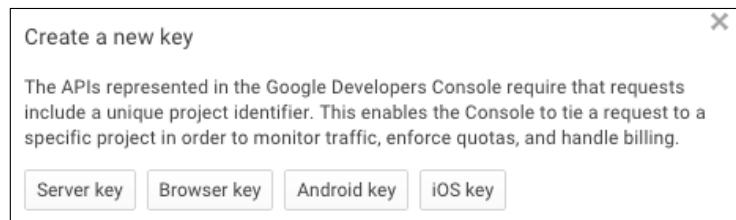
5. Go to **Credentials** to create your own key:



6. Click on the **Create new Key** button under the **Public API Access** section:



7. Click on the **iOS key** button:



## *Adding Device Features Support* —

8. Fill in your app's Bundle ID. You might not know exactly what it is yet because Ionic will create a random ID. So just put in `com.ionicframework.starter` and change that later.

Create an iOS key and configure allowed iOS applications

This key can be deployed in your iOS applications.

API requests are sent directly to Google from your clients' iOS devices. Google verifies that each request originates from an iOS application that matches one of the bundle identifiers listed below. [Learn more](#)

Accept requests from an iOS application with one of the bundle identifiers listed below (Optional)  
One bundle identifier per line. Example: com.example.MyApp  
Or if you leave this blank, requests will be accepted from any iOS application. Be sure to add bundle identifiers before using this key in production.

**Create** **Cancel**

9. Click on the **Create** button.

10. Now you should see the **Key for iOS applications** section like the following:

Key for iOS applications	
API key	
iOS applications	com.ionicframework.starter com.ionicframework.googlemaps215005
Activation date	Jul 17, 2014, 5:40:00 PM
Activated by	quanghoc@gmail.com (you)
<b>Edit allowed iOS applications</b> <b>Regenerate key</b> <b>Delete</b>	

11. Copy that API key so you can use it to add the Cordova Google Maps plugin.

## **How to do it...**

Let's start an Ionic project from scratch and add Google Maps features:

1. Create a blank Ionic project and go to that folder:

```
$ ionic start GoogleMaps blank  
$ cd GoogleMaps
```

2. Install the Google Maps plugin with your copied key replacing YOUR\_IOS\_API\_KEY\_IS\_HERE:

```
$ cordova plugin add plugin.google.maps --variable API_KEY_FOR_IOS="YOUR_IOS_API_KEY_IS_HERE"
```

If you do this for both iOS and Android, use the following command line:

```
$ cordova plugin add plugin.google.maps --variable API_KEY_FOR_ANDROID="key" --variable API_KEY_FOR_IOS="key"
```

3. Since the Google Maps plugin is not part of the ngCordova support, there is no need to install ngCordova in this project. Let's open index.html to start working on the view.
4. The first thing you may want to do is to disable telephone number detection in iOS. The reason is that some of the street address matches the phone number format and it may interfere with your model binding. So add the following meta tag in your index.html header:

```
<meta name="format-detection" content="telephone=no">
```

5. Your UI will consist of a top header bar with one button and the rest of the screen will be Google Maps. You need to assign GoogleMapsCtrl first.

```
<body ng-app="starter" ng-controller="GoogleMapsCtrl">
  <ion-pane>
    <ion-header-bar align-title="left" class="bar-stable"
      ng-click="setCenterLocation()">
      <h1 class="title">{{ data.address }}</h1>
      <button class="button ion-location button-balanced"
        ng-click="gotoMyLocation()"></button>
    </ion-header-bar>
    <ion-content class="full-height">
      <ion-map width="100%" height="100%" map="map">
      </ion-map>
    </ion-content>
  </ion-pane>
</body>
```

If the user clicks on the header bar, it will get the middle coordinate of the map and return the address at that location. You will write that functionality in setCenterLocation() later. The gotoMyLocation() function basically gets the current location of the device and shows a marker there with longitude and latitude. The most important component here is <ion-map>, for which you will create a directive called ionMap. The idea is to call the Google Maps plugin to render the map inside this DOM object.

6. Ionic does not have full screen by default, so you need to customize the style sheet by adding a new class. Let's open `/www/css/style.css` to add a class:

```
.full-height .scroll {  
    height: 100%;  
}
```

7. Now open `app.js` and start to create a controller, a directive, and a factory:

```
var app = angular.module('starter', ['ionic'])  
app.controller('GoogleMapsCtrl', function($scope,  
$ionicPlatform, $ionicMap) {  
});  
app.directive("ionMap", function ($ionicPlatform,  
$ionicMap) {  
});  
app.factory('$ionicMap', function($ionicPlatform) {  
});
```

The controller will only have two functions: `gotoMyLocation()` and `setCenterLocation()`, so let's cover that one last. Most of the critical code will be `$ionicMap` as that is your assigned delegate to provide many *map services*.

8. You need to make objects private to keep track of the map object and current center location when the user moves the map around. Put these two lines right under the `$ionicMap` factory:

```
var map = {}  
centerLoc = {};
```

9. This `$ionicMap` factory also returns a list of public functions. The skeleton looks like this:

```
return {  
    init: function(div) {  
    },  
    getMap: map,  
    gotoMyLocation: function() {  
    },  
    setCenterLocation: function(callback) {  
    }  
}
```

You will start writing code for each of the functions.

10. Let's start with the `init()` function, which is when the map is initialized from the plugin.

```
init: function(div) {
  $ionicPlatform.ready(function() {
    // Initialize the map view
    map = plugin.google.maps.Map.getMap(div);

    // Wait until the map is ready status.
    map.addEventListener(plugin.google.maps.event.MAP_READY, function() {
      console.log('MAP_READY');
    });

    // When map is moved, get new center location
    map.on(plugin.google.maps.event.CAMERA_CHANGE,
    function(position) {
      centerLoc = position;
    });
  });
  return;
},
```

It's important to call `plugin.google.maps.Map.getMap` on some `div` element because the plugin needs to know where it should render the map. You will call this `init()` function from the directive later. There is even a camera change called `plugin.google.maps.event.CAMERA_CHANGE` so that each time the user moves the map around, you can get the center location and do this assignment: `centerLoc = position`.

11. For the button to trigger the map to move to your location, you can use the `map.getMyLocation()` function of the plugin.

```
gotoMyLocation: function() {

  // Get current coordinate where you are located
  map.getMyLocation(function(location) {
    var msg = ["Current your location:\n",
      "latitude:" + location.latLng.lat,
      "longitude:" + location.latLng.lng].join("\n");

    // Move the camera to your location
  });
}
```

```
    map.moveCamera({
        'target': location.latLng,
        'zoom': 15
    });

    // Add a marker with position and title text
    map.addMarker({
        'position': location.latLng,
        'title': msg
    }, function(marker) {
        marker.showInfoWindow();
    });
});

},
});
```

The location object returned will contain latitude (`location.latLng.lat`) and longitude (`location.latLng.lng`). To move the camera to anywhere, just call `map.moveCamera` by passing the location coordinate (`location.latLng`). To add a marker, call `map.addMarker` with the position and title as HTML. That's it.

12. The `setCenterLocation()` function performs multiple functions at once. First, it needs to use the `centerLoc` object to add a marker right in the middle of the map. Second, from the coordinate data, Google geocoder will return the official address. Here is the code:

```
setCenterLocation: function(callback) {
  if (centerLoc.hasOwnProperty('target')) {
    var msgCenter = [
      "latitude:" + centerLoc.target.lat,
      "longitude:" + centerLoc.target.lng].join("\n");

    map.addMarker({
      'position': centerLoc.target,
      'title': msgCenter
    }, function(marker) {
      marker.showInfoWindow();
    });

    var request = {
      'position': centerLoc.target
    };

    // Passing longitude and latitude and get back
    the address
```

```
plugin.google.maps.Geocoder.geocode(request,
function(results) {
    if (results.length) {
        var result = results[0];
        var position = result.position;
        var address = [
            result.thoroughfare || "",
            result.locality || "",
            result.adminArea || "",
            result.postalCode || "",
            result.country || ""].join(", ");
        // Trigger callback function to provide the
        // address string to the controller
        callback(address);
    } else {
        console.log("Not found");
    }
});
} else {
    console.log("No location defined");
}
})
```

Note that for `plugin.google.maps.Geocoder.geocode` to work, you must pass the request object with the `position` key. If you send `centerLoc.target` as the parameter, it will not work. Once the address is returned, it will trigger the `callback` parameter to pass the address string: `callback(address)`.

13. Now that you have the `$ionicMap` delegate defined, this will be your `<ion-map>` directive:

```
app.directive("ionMap", function ($ionicPlatform,
$ionicMap) {
    return {
        restrict: 'AEC',
        link: function(scope, element, attrs) {
            $ionicPlatform.ready(function() {

                // Create a div object
                var div = document.createElement('div');
                div.style.width = attrs.width;
                div.style.height = attrs.height;

                // Add this div to the DOM
```

```
        element.append(div);

        // Turn the div into Google Maps object
        $ionicMap.init(div);
    });

}

});
});
```

AngularJS automatically translates the directive naming of `ionMap` into the `<ion-map>` tag in the HTML code. The `attrs` object will contain all the attributes in this tag. So to reference an attribute, you can just call `attrs.width` or `attrs.height`. The goal of this directive is to pass the width and height to this new `div` element and append it in the DOM. Then it would call the `$ionicMap` delegate to initialize the map right on this `div` element.

14. The last part is the easiest component to write. It will be your controller:

```
app.controller('GoogleMapsCtrl', function($scope,
$ionicPlatform, $ionicMap) {
    $scope.data = {
        address: "Tap for address"
    }

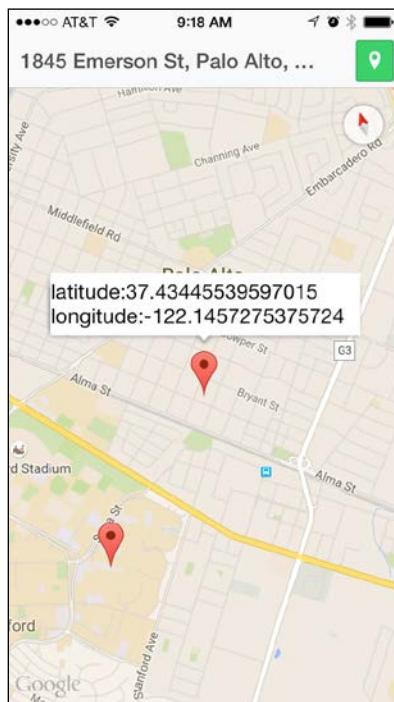
    $scope.gotoMyLocation = $ionicMap.gotoMyLocation;
    $scope.setCenterLocation = function() {
        $ionicMap.setCenterLocation(function(address) {
            $scope.data.address = address;

            // Call digest cycle to update $scope.data.address
            $scope.$digest();
        });
    }
});
```

15. If you run the app in your mobile device (not in the browser as the plugin does not work locally), you should be able to see something like the following screenshot:



16. Tap the green button for the current location. Tap the top header bar to identify the current address in the middle of the map.



## How it works...

The `gotoMyLocation()` function is a straight-up assignment. The `setCenterLocation()` function will get the address within the callback of `$ionicMap.setCenterLocation`. It's important that you trigger AngularJS' digest cycle in the callback because AngularJS will not have knowledge of the variable change when it's outside the controller cycle.

This is to ensure the address change will be updated in the view as well. The `$scope.$digest()` function is just a way to tell AngularJS to re-scan all variables for change detection. If their current values are different from their previous values, then update them in the `$scope` object. By default, Angular triggers the digest cycle in each controller and directive. This *cycle* will do two main things:

- ▶ Detect changes in scope variables
- ▶ Update the changes in the UI or other bindings

Although it might seem like there are a lot of moving parts, the basic flow is very simple:

1. Whenever Ionic and Cordova are ready, trigger a callback of `$ionicPlatform.ready` to initialize the map.
2. Create a `div` element and append into the `ion-map` DOM object.
3. Call `getMyLocation` to get location data.
4. If the map is moved, trigger the event `map.on(plugin.google.maps.event.CAMERA_CHANGE,...)` and update the middle coordinate.
5. Call `plugin.google.maps.Geocoder.geocode` to get the address string of the coordinate.

It's important to know that `plugin.google.maps.Map.getMap` does take some time to process, and it will trigger a `ready` event once it has successfully created the map. That's why you need to add an event listener for `plugin.google.maps.event.MAP_READY`. This example does not do anything right after the map is ready, but in the future you could add more processing functions such as to jump to the current location automatically or add more markers on the top of the map.

### There's more...

The Cordova Google Maps plugin has many more features such as:

- ▶ Show an InfoWindow
- ▶ Add a marker with multiple lines
- ▶ Modify an icon
- ▶ Text styling
- ▶ Base64 encoded icon
- ▶ Click on a marker
- ▶ Click on an InfoWindow
- ▶ Create a draggable marker
- ▶ Drag events
- ▶ Create a flat marker

Since you cannot pop up a `div` element on top of the native Google Maps, the Marker features are very handy. There are some additional scenarios:

- ▶ Touch a marker and go to a page: you just need to listen to the `plugin.google.maps.event.MARKER_CLICK` event and do whatever is needed in the callback function.
- ▶ Show an avatar / profile image as a marker: `addMarker` does take the base64 image string. So you can pass something like this in the argument `title: canvas.toDataURL()`.

Note that Google has a quota on free API usage. For example, you cannot exceed one request per second per use, and you can only have a couple of thousand requests per day. This quota changes all the time, but it's important to know about it. In any case, if you have problems with your key, you have to go back to the **Credentials** page and regenerate the key. In order to change the key manually in your app, you have to edit `/plugins/ios.json`. Look for two places. The first one is this:

```
"*-Info.plist": {  
    "parents": {  
        "Google Maps API Key": [  
            {  
                "xml": "<string>YOUR_IOS_API_KEY_IS_HERE</string>",  
                "count": 1  
            }  
        ]  
    }  
}
```

And this is the second one:

```
"plugin.google.maps": {  
    "API_KEY_FOR_IOS": "YOUR_IOS_API_KEY_IS_HERE",  
    "PACKAGE_NAME": "com.ionicframework.starter"  
}
```

You just need to edit the `YOUR_IOS_API_KEY_IS_HERE` line and replace it with your new key.

## See also

There are lots of ways to work with Google Maps. You can visit the GitHub page of the Google Maps plugin to learn more at <https://github.com/wf9a5m75/phonegap-googlemaps-plugin>.

# 4

## Offline Data Storage

In this chapter, we will cover the following tasks related to persistent data operations:

- ▶ Creating a to-do app using ngStorage for Local Storage
- ▶ Creating a social networking app using SQLite

### Introduction

In this chapter, you will learn how to store and retrieve data from a local device when there is no Internet access. This is important for some apps because of their specific use cases. For example:

- ▶ The data does not need to be stored in the server
- ▶ It's faster to store data locally and sync it to the server using background processing
- ▶ The app must work locally and send data to the server when it's online again

There are several methods for offline data persistence.

- ▶ In the past, you probably leveraged cookies in some aspects to store session data such as information regarding whether a user has visited a certain page. Cookies are used for very basic use cases because of their size limitation (4,095 bytes). Cookies are sent to a server on each HTTP request so that the server also has access to this information. This may result in a waste of bandwidth.
- ▶ Browsers today have Local Storage support, which is a better method. Local Storage can store up to 5 MB of data and is available only to the client. There are also various libraries supporting Local Storage in the open source community.
- ▶ SQLite is also a great alternative for offline data access. However, SQLite is only available as a device-specific feature and not as JavaScript-only. Therefore, in order to use SQLite, you must install a Cordova plugin to support this feature.

In general, you can use the following table to look at criteria, to determine the right technology to use for your offline data storage:

Criteria	Cookie	Local Storage	SQLite
Data available on the server side	Yes	No	No
Data size	<4 KB	<5 MB	This depends on the device space (can be in terabytes)
JavaScript-only	Yes	Yes	The Cordova plugin is required
Older device support	Yes	Yes	You may see performance issues
Complex many-to-many relationships	No	You may see performance issues	Yes

This chapter will go into detail regarding how to work with both Local Storage and SQLite.

## Creating a to-do app using ngStorage for Local Storage

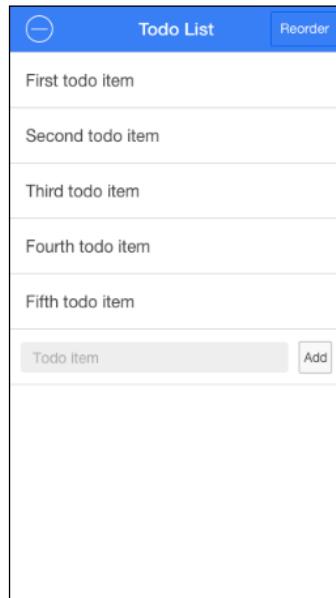
There are many to-do apps out there, but it could get very complex when you just want to explore one capability of the platform or framework. The example in this recipe will leverage the idea of a to-do app to explain how Local Storage works. Most apps use Local Storage as a method to ensure that the app data is persistent and available offline. You will learn some basic knowledge about how to manipulate data by adding, editing, and deleting them. These are some very common operations for any app.

Here is the high-level process:

- ▶ Initialize a Local Storage object with some sample data
- ▶ Render an array of objects in the frontend
- ▶ Allow the user to delete an item in the array and then update Local Storage
- ▶ Allow the user to edit an item in the array by using \$ionicPopup
- ▶ Allow the user to move the position of an item by manipulating the Local Storage array

You will not access the `localStorage` object directly. Instead, you will use an existing module called `ngStorage`. This module is compatible with AngularJS and works very well in an Ionic app.

The following is a screenshot of the to-do app that you are going to build:



## Getting ready

You don't need to test Local Storage on a physical device, because it will work in a browser. If you want to test the offline capability, you can even turn off the network connectivity and the app should still be functional.

## How to do it...

Follow these steps to create a to-do app:

1. Start a blank project (for example, LocalStorage) and go to that folder, as follows:

```
$ ionic start LocalStorage blank  
$ cd LocalStorage
```

2. Install ngStorage:

```
$ bower install ngstorage
```

3. Add ngStorage in the header above app.js:

```
<script src="lib/ngstorage/ngStorage.js"></script>
```

4. Add a controller in the `<body>` tag or a parent tag:

```
<body ng-app="starter" ng-controller="MainCtrl">
```

The body will consist of the header bar and content area. The header bar will show the **Delete** and **Reorder** button. The content area will show your to-do list.

5. Create a header bar within the `<body>` tag:

```
<ion-header-bar class="bar-positive">
  <div class="buttons">
    <button class="button button-icon icon ion-ios-minus-outline"
      ng-click="data.showDelete = !data.showDelete;
      data.showReorder = false"></button>
  </div>
  <h1 class="title">Todo List</h1>
  <div class="buttons">
    <button class="button" ng-click="data.showDelete =
      false; data.showReorder = !data.showReorder">
      Reorder
    </button>
  </div>
</ion-header-bar>
```

This header bar is very simple. Both the **Delete** and **Reorder** buttons are *wired* with some variables. Delete and reorder are two different *modes* that the user can do only once at a time. The two Boolean variables, `data.showDelete` and `data.showReorder`, will be passed on to the `<ion-list>` directive later as a *toggle* for the action.

6. Now, let's add the `<ion-content>` tag to display the to-do list:

```
<ion-content>
  <ion-list show-delete="data.showDelete"
    show-reorder="data.showReorder">

    <ion-item ng-repeat="item in $storage.items"
      item="item" class="item-remove-animate">
      {{ item.label }}
      <ion-delete-button class="ion-minus-circled"
        ng-click="onItemDelete(item)">
      </ion-delete-button>
      <ion-option-button class="button-assertive"
        ng-click="edit($index)">
        Edit
      </ion-option-button>
    </ion-item>
  </ion-list>
</ion-content>
```

```

<ion-reorder-button class="ion-navicon"
on-reorder="moveItem(item, $fromIndex,
$toIndex)"></ion-reorder-button>
</ion-item>
<div class="item item-input-inset">
<label class="item-input-wrapper">
<input type="text" placeholder="Todo item"
ng-model="data.item">
</label>
<button class="button button-small"
ng-click="addItem()">
    Add
</button>
</div>

</ion-list>
</ion-content>

```

 The last row will allow the user to add a new to-do item. This is wrapped within the `<div class="item item-input-inset">` element. Each `<ion-item>` element is just a `div` element with classes. In this case, you shouldn't use the `<ion-item>` directive itself, because you need to customize the internal components. This last item will contain only an input box and a button.

7. The next step is to create a controller to handle all the events. Open `app.js` to modify its content by first adding `ngStorage` as the dependency:

```
var app = angular.module('starter', ['ionic',
'ngStorage']);
```

8. The `MainCtrl` controller will need to have three dependencies: `$ionicPopup` to handle the edit popup, `$ionicListDelegate` to let you trigger the close of the **Option** button, and `$localStorage` from `ngStorage`:

```
app.controller('MainCtrl', function($scope, $ionicPopup,
$ionicListDelegate, $localStorage) {
});
```

9. First, you need to initialize this controller, as follows:

```
$scope.$storage = $localStorage.$default({
  items: [
    { label: 'First todo item' },
    { label: 'Second todo item' },
    { label: 'Third todo item' },
  ]
});
```

```
{ label: 'Fourth todo item' },
{ label: 'Fifth todo item' }
]
});
```



There is no need to initialize `showDelete` or `showReorder` because by default, we want them to be false anyway. In JavaScript, an undefined variable is false.

To use Local Storage, you just need to assign a scope variable to `$localStorage`. `ngStorage` will take care of the binding and updating of the `$scope` object. There is no need to call `getter()` or `setter()` as you may have seen in some other solutions. To assign a default value, you just need to call `$default()`. You can treat Local Storage like a NoSQL solution, as you can just store any object with any structure.

10. To edit an item, the controller will call `$ionicPopup` to show a popup with one field to edit:

```
$scope.edit = function(index) {
  $scope.editItem = {label:
    $scope.$storage.items[index].label};
  var itemPopup = $ionicPopup.show({
    template: '<input type="text"
ng-model="editItem.label">',
    title: 'Edit Todo',
    scope: $scope,
    buttons: [
      { text: 'Cancel' },
      {
        text: '<b>Save</b>',
        type: 'button-positive',
        onTap: function(e) {
          if (!$scope.editItem.label) {
            e.preventDefault();
          } else {
            $scope.$storage.items[index].label =
              $scope.editItem.label;
          }
        }
      }
    ]
  });
}
```

```
        return $scope.editItem;
    }
}
]
});
itemPopup.then(function(res) {
  $ionicListDelegate.closeOptionButtons();
});
};
```

In the view, when the **Edit** button is clicked, you also pass the index of the current item in the `ng-click="edit($index)"` array. By doing so, you can access the *label* of that object by calling `$scope.$storage.items[index].label`. That way, when the popup appears, it will show the value of the current item, which will allow the user to edit the item.

11. To move or reorder an item, the `moveItem()` function will manipulate the array:

```
$scope.moveItem = function(item, fromIndex, toIndex) {
  $scope.$storage.items.splice(fromIndex, 1);
  $scope.$storage.items.splice(toIndex, 0, item);
};
```

The reordering action is done by using an array's `splice()` function, which is basically a simple *trick* that can be used to delete an array item and insert it back in a different index.

12. Next, to delete an item, the `onItemDelete()` function will handle `splice()` of the exact item index:

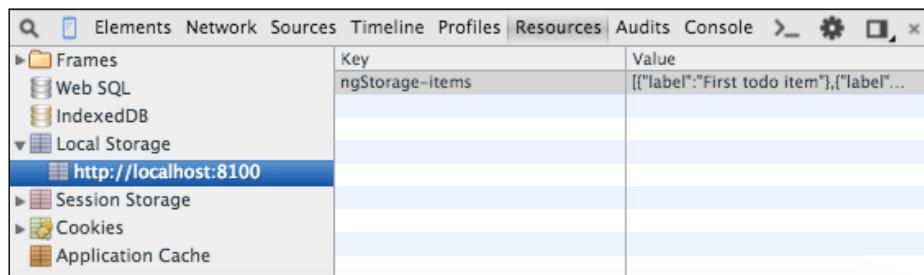
```
$scope.onItemDelete = function(item) {
  $scope.$storage.items.splice($scope.$storage.
    items.indexOf(item), 1);
};
```

13. Finally, to add an item, you can just push the object into the `$storage.items` array:

```
$scope.addItem = function() {
  $scope.$storage.items.push({label: $scope.data.item});
  $scope.data.item = '';
}
```

After it's pushed, just reset `$scope.data.item` to an empty string again.

14. That's it. If you test the app in the browser via `ionic serve`, you should be able to see the Local Storage object being stored. Navigate to the **Resources** tab, expand **Local Storage**, and click on `http://localhost:8100/`, as follows:



## How it works...

This recipe introduced the usage of `<ion-list>`. This directive has two attributes: `show-delete` and `show-reorder` in the view. If the value is `true`, it will enable that feature. If you pass a variable to these attributes, they can be turned on and off programmatically. Under `<ion-list>`, there are two components: a list of the existing to-do items from Local Storage and an input box that can be used to add a new to-do item.

You can loop through a Local Storage object (`$storage.items`) just like any `$scope` object. The `$storage` object defined in the controller is basically a Local Storage object created by `ngStorage`, which makes it a lot easier to manipulate variables in an AngularJS way. To ensure that there are some elements that can be used to handle delete and reorder, you need to put the `<ion-delete-button>` and `<ion-reorder-button>` directives within `<ion-item>`. This includes an individual button for each item. Besides these *special function* buttons, you can add custom buttons using `<ion-option-button>`. All of these buttons must of course be assigned with `ng-click` to process the event.

In this recipe, `$ionicPopup` was also introduced. This is an out-of-the-box Ionic delegate, where you can control a custom pop-up dialog. When you click on the popup's button, `onTap` will be triggered. You don't want the user to close and save the to-do item as an empty string. That's why you need to call `e.preventDefault()` in order to ignore the tap event. Otherwise, you need to reassign the new label to `$scope.$storage.items[index].label`. In addition to this, the reason behind why you need to call `itemPopup.then()` is that when you click on the option of the `<ion-item>`, it doesn't close automatically. Therefore, you need to close this button once the user returns from the popup.

Looking under the hood, ngStorage just leverages the native `window.localStorage` object. There is no cookie failover, because most browsers already have support for the Local Storage technology. ngStorage abstracts all the steps to convert JSON to a string and vice versa. It also sets up `$watch` for the `$scope` object to detect changes in the Local Storage objects. That's why you can use `$scope.$storage` in the same way as the normal AngularJS `$scope` variables.

### There's more...

For iOS devices, the data in Local Storage will be backed up to iCloud automatically by default. This means that the app data will be persistent even if the user updates the app or restores the entire iPhone. To disable automatic backup, you need to modify the `config.xml` of your Ionic project, as follows:

```
<preference name="BackupWebStorage" value="none" />
```

If you want to completely erase all the Local Storage data of your app, use the `$reset()` function, as follows:

```
$localStorage.$reset();
```

### See also

For more information about ngStorage, you can access their GitHub project at <https://github.com/gsklee/ngStorage>.

## Creating a social networking app using SQLite

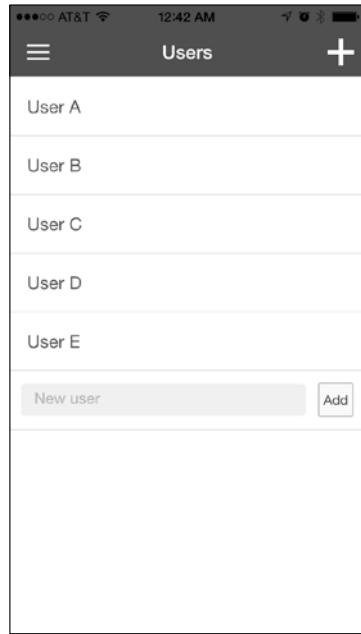
This recipe will discuss how you can build a complex relational database app using SQLite. In a real-world scenario, the database usually stays in the backend, where it's more feasible to process a large amount of data for multiple users.

However, there are some cases where the app just needs to work on a limited set of data in the local client device. While it's possible to leverage Local Storage, you will run into the 5 MB limitation of Local Storage very quickly. In addition, you can take advantage of the relational database features, such as table joins, because such a query can have a very poor performance using Local Storage.

## *Offline Data Storage*

---

It's easy to get started by using a very common example in most social networking apps, where there are definitions of users and memberships. You will go through a process in this recipe for the creation of an app to allow users to *subscribe* to various *groups*. This is very similar to how Facebook or Twitter may work. Here is a screenshot of the app that you will create:



You will learn how to:

- ▶ Create a navigation menu and routes
- ▶ Add an item and save it in the SQLite database
- ▶ Update an item
- ▶ Query an item by joining multiple tables (that is, users and groups)

### **Getting ready**

Since SQLite does not work in the browser, you have to test the app using a real mobile device or simulator.

## How to do it...

Here are the instructions to create a social networking app using SQLite:

1. Start a blank project (for example, SQLite) and go to that folder:

```
$ ionic start SQLite blank  
$ cd SQLite
```

2. Install ngCordova, as follows:

```
$ bower install ngCordova
```

3. Open index.html and add ngCordova in the header above cordova.js:

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
```

4. Install the Cordova SQLite plugin:

```
$ cordova plugin add https://github.com/litehelpers/Cordova-sqlite-storage.git
```

5. Change the body tag of index.html to the following:

```
<body ng-app="starter">  
  <ion-nav-view></ion-nav-view>  
</body>
```

This app will leverage the sidemenu layout. That's why you just need to start with `<ion-nav-view>`, as the rest will be injected into that directive.

6. Create a layout.html file and save it in the templates folder under www:

```
<ion-side-menus enable-menu-with-back-views="true">  
  <ion-side-menu-content>  
    <ion-nav-bar class="bar bar-dark">  
      <ion-nav-buttons side="left">  
        <button class="button button-icon icon ion-navicon" menu-toggle="left"></button>  
      </ion-nav-buttons>  
    </ion-nav-bar>  
  
    <ion-nav-view name="main" animation="slide-left-right"></ion-nav-view>  
  </ion-side-menu-content>  
  
<ion-side-menu side="left" width="150">
```

```
<ion-content>
  <ion-list>
    <ion-item class="item" ui-sref="app.users"
      menu-close>
      Users
    </ion-item>
    <ion-item class="item" ui-sref="app.groups"
      menu-close>
      Groups
    </ion-item>
  </ion-list>
</ion-content>
</ion-side-menu>

</ion-side-menus>
```

This layout will give you `<ion-nav-view>` as the content area. The menu will be on the left, with **Users** and **Groups** as two menu items.

7. The **Users** menu will let you view a list of users. If you click on a user, the app will show a modal with the groups that the user belongs to. You can save the changes in this modal and review them in the **Groups** menu. This **Groups** page is just a view-only page that provides a summary of the mapping relationship between the users and groups. So first, let's create a `users.html` template:

```
<ion-view view-title="Users">
  <ion-content>
    <ion-list>
      <ion-item class="item" ng-repeat="user in users"
        ng-click="openUserModal(user)">
        {{ user.name }}
      </ion-item>
      <div class="item item-input-inset">
        <label class="item-input-wrapper">
          <input type="text" placeholder="New user"
            ng-model="newUser.name">
        </label>
        <button class="button button-small"
          ng-click="addUser()">
          Add
        </button>
      </div>
    </ion-list>
  </ion-content>
</ion-view>
```

Note that the last line item is used to add a new user. This function is called `adduser()`. You will write a controller for this later. Also, another function that you should take note of is `openUserModal(user)`, which will list all the groups the user belongs to.

8. For the user modal that shows the group list, you have to create a separate file called `userModal.html`:

```
<ion-modal-view>
  <ion-header-bar class="bar bar-dark">
    <button class="button button-clear"
      ng-click="cancel()">Cancel</button>
    <h1 class="title">User Detail</h1>
    <button class="button button-clear"
      ng-click="save(user, usergroup)">Save</button>
  </ion-header-bar>
  <ion-content>
    <div class="list">
      <label class="item item-checkbox" ng-repeat="group
        in groups">
        <label class="checkbox">
          <input type="checkbox"
            ng-model="usergroup[$index]">
        </label>
        {{ group.name }}
      </label>
    </div>
  </ion-content>
</ion-modal-view>
```

This modal just has the **Cancel** and **Save** buttons. The group list is just shown as checkboxes in `ng-repeat` of the `$scope.groups` model. You will also write the controller function to handle this later.

9. The final template is used to display the content in the **Groups** menu. Let's call this the `groups.html` file:

```
<ion-view view-title="Groups">
  <ion-nav-buttons side="right">
    <button class="button button-icon button-clear
      ion-plus" ng-click="openGroupModal()"/></button>
  </ion-nav-buttons>
  <ion-content>
    <div class="list">
      <div ng-repeat="group in groups">
        <label class="item item-divider">
```

```
        {{ group.name }}  
    </label>  
    <label class="item" ng-repeat="user in  
group.users">  
        {{ user.name }}  
    </label>  
    </div>  
    </div>  
  </ion-content>  
</ion-view>
```

Like the group list modal, this one also lists all the groups, including the users in each group. You can get an overall view of the relationship of the group and its members. This is a view-only template, as there won't be any function to handle change or save. The purpose is to show how the SQLite query join works.

10. Now, let's move to `app.js`, where you will write the controller and services to handle all the data operations. First, you need to create the routes, as follows:

```
var app = angular.module('starter', ['ionic',
'ngCordova']);

app.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    .state('app', {
      url: "/app",
      abstract: true,
      templateUrl: "templates/layout.html",
      controller: 'MainCtrl'
    })
    .state('app.users', {
      url: "/users",
      views: {
        'main': {
          templateUrl: "templates/users.html",
          controller: "UsersCtrl"
        }
      }
    })
    .state('app.groups', {
      url: "/groups",
      views: {
        'main': {
          templateUrl: "templates/groups.html"
        }
      }
    })
});
```

```
        }
    }
}) ;

$urlRouterProvider.otherwise('/app/users');
});
```

The routing is straightforward, as it will start with `layout.html` and replace the `<ion-nav-view name="main">` node with either `users.html` or `groups.html`, depending on the URL. Note that you need to declare `ngCordova` as a dependency for the app.

11. In order to handle the database, you need to create a factory to open the database, populate it with initial data, and perform data operations specifically for this app. Let's add the following to `app.js`:

```
app.factory('MyData', function($ionicPlatform,
$cordovaSQLite, $q) {
    var db = {},
        users = [],
        groups = [],
        usergroup = [];

    var initdb = {
        users: ["User A", "User B", "User C", "User D",
        "User E"],
        groups: ["Group 1", "Group 2", "Group 3"]
    };
}) ;
```

You will use the `db` object to store the database object itself, which is basically a local file in the device filesystem. The `users` and `groups` will keep track of the existing data in the memory for **Users** and **Groups**. In order to map the relationship between them, you will use the `usergroup` array. You can change the values in `initdb`, as it's used to initialize the database with some sample datasets.

12. When the app is started, you need to create the tables (this example actually drops the previous tables to start afresh) and populate them with `initdb`:

```
var createUsers = function() {
    $cordovaSQLite.nestedExecute(db,
        'CREATE TABLE IF NOT EXISTS users (id integer primary
key, name text)',
        'INSERT INTO users (name) VALUES (?, ?, ?, ?, ?)', [
        ],
        initdb.users
```

```
        ) .then(function(res) {  
  
    $cordovaSQLite.execute(db, 'SELECT * FROM  
users') .then(function(res) {  
  
        for (var i=0; i<res.rows.length; i++) {  
            users.push(res.rows.item(i));  
        }  
  
    }, function (err) {  
        console.error(err);  
    });  
  
}, function (err) {  
    console.error(err);  
});  
};  
  
var createGroups = function() {  
    $cordovaSQLite.nestedExecute(db,  
        'CREATE TABLE IF NOT EXISTS groups (id integer primary  
key, name text)',  
        'INSERT INTO groups (name) VALUES (?,?,?)',  
        [],  
        initdb.groups  
    ) .then(function(res) {  
  
        $cordovaSQLite.execute(db, 'SELECT * FROM  
groups') .then(function(res) {  
  
            for (var i=0; i<res.rows.length; i++) {  
                groups.push(res.rows.item(i));  
            }  
  
        }, function (err) {  
            console.error(err);  
        });  
  
    }, function (err) {  
        console.error(err);  
    });  
}  
};  
  
$ionicPlatform.ready(function() {  
    //...  
});
```

```

db = $cordovaSQLite.openDB("my.db", 1);

$cordovaSQLite.execute(db, 'DROP TABLE IF EXISTS
users').then(function(res) {
    createUsers();
}, function (err) {
    console.error(err);
});

$cordovaSQLite.execute(db, 'DROP TABLE IF EXISTS
groups').then(function(res) {
    createGroups();
}, function (err) {
    console.error(err);
});

$cordovaSQLite.execute(db, 'DROP TABLE IF EXISTS
usergroup').then(function(res) {
    $cordovaSQLite.execute(db, 'CREATE TABLE IF NOT EXISTS
usergroup (id integer primary key, userId integer,
groupId integer)').then(function(res) {
    }, function (err) {
        console.error(err);
    });

}, function (err) {
    console.error(err);
});
});

}) ;

```

This initialization code must be inside the factory and not in the `return` object because we want it to be executed once the factory name is declared in the controller. This app will start afresh with no *mapping relationship* between any user and group. So, your `usergroup` table will be empty at the beginning.

13. You can continue to write the rest of the functions that handle data in the `return {}` function of the `MyData` factory, as follows:

```

return {
    users: users,
    groups: groups,
    getGroupsByUserId: function(userId) {
        var q = $q.defer();
        var query = "SELECT groupId FROM usergroup WHERE
userId = (?)";

```

```
$cordovaSQLite.execute(db, query,
  [userId]).then(function(res) {
  q.resolve(res);
}, function (err) {
  console.error(err);
  q.reject(err);
});

return q.promise;
},
getGroupsAll: function() {
  var q = $q.defer();
  var query = "SELECT groups.id, groups.name,
  GROUP_CONCAT(usergroup.userId) AS userIds FROM groups
  LEFT OUTER JOIN usergroup ON groups.id =
  usergroup.groupId GROUP BY usergroup.groupId";
  $cordovaSQLite.execute(db, query).then(function(res) {
    q.resolve(res);
  }, function (err) {
    q.reject(err);
  });
}

return q.promise;
},
addUser: function(params) {
  var q = $q.defer();
  var query = "INSERT INTO users (name) VALUES (?)";
  $cordovaSQLite.execute(db, query,
  [params.name]).then(function(res) {
    q.resolve(res);
  }, function (err) {
    console.error(err);
    q.reject(err);
  });
}

return q.promise;
},
updateGroupByUserId: function(userId, usergroups) {
  var q = $q.defer();
  var query = "DELETE FROM usergroup WHERE userId = (?)";
  $cordovaSQLite.execute(db, query,
  [userId]).then(function(res) {
```

```
var query = "INSERT INTO usergroup (userId, groupId)
VALUES (?,?)";
$cordovaSQLite.insertCollection(db, query,
usergroups).then(function(res) {
    q.resolve(res);
}, function (err) {
    console.error(err);
    q.reject(err);
});

}, function (err) {
    console.error(err);
    q.reject(err);
});

return q.promise;
}
}
```

14. The next step is to create the UsersCtrl controller:

```
app.controller('UsersCtrl', function($scope, $timeout,
$ionicModal, $cordovaSQLite, MyData) {
    $scope.newUser = {};

    $ionicModal.fromTemplateUrl('templates/userModal.html', {
        scope: $scope,
        animation: 'fade-in'
    }).then(function(modal) {
        $scope.userModal = modal;
    });

    $scope.openUserModal = function(user) {
        $scope.user = user || {};
        $scope.usergroup = [];
        if ((user) && (angular.isObject(user)) &&
(user.hasOwnProperty('id'))) {
            $scope.user.groups = [];
            MyData.getGroupsByUserId(user.id).then(function(res)
{
                for (var i=0; i<res.rows.length; i++) {
                    $scope.user.groups.push(
                    res.rows.item(i).groupId);
                }
            })
        }
    }
});
```

```
        for (var i=0; i<$scope.groups.length; i++) {
            $scope.usergroup.push(
                $scope.user.groups.indexOf(
                    $scope.groups[i].id) >= 0);
        }
    });
}

$scope.userModal.show();
};

$scope.addUser = function() {
    MyData.addUser($scope.newUser.name).then(function(res) {
        $scope.users.push({
            id: res.insertId,
            name: $scope.newUser.name
        });
        $scope.newUser.name = '';
    });
}

$scope.save = function(user, usergroup) {
    var usergroups = [];
    for (var i=0; i<$scope.groups.length; i++) {
        if (usergroup[i]) {
            usergroups.push([user.id, $scope.groups[i].id]);
        }
    }

    MyData.updateGroupByUserId(user.id,
        usergroups).then(function(res) {
    });

    $scope.userModal.hide();
};

$scope.cancel = function() {
    $scope.userModal.hide();
};

}) ;
```

15. The following is the code that is used to handle MainCtrl, which tights to the app state. If you assign a controller to a higher-level state, it will automatically become the parent controller of its children states' controllers:

```
app.controller('MainCtrl', function($scope, $rootScope,
MyData) {
    $scope.users = MyData.users;
    $scope.groups = MyData.groups;

    $rootScope.$on('$stateChangeStart', function(event,
toState, toParams, fromState, fromParams) {
        if (toState.name == 'app.groups') {
            getGroups();
        }
    });

    function getGroups() {
        MyData.getGroupsAll().then(function(res) {
            var newGroups = [];
            for (var i=0; i<res.rows.length; i++) {
                newGroups[i] = {
                    id: res.rows.item(i).id,
                    name: res.rows.item(i).name,
                    users: []
                };
                var userIds = res.rows.item(i).userIds ?
res.rows.item(i).userIds.split(',') : [];
                for (var j=0; j<userIds.length; j++) {
                    var name = '';
                    for (var t=0; t<$scope.users.length; t++) {
                        if ($scope.users[t].id == userIds[j])
                            name = $scope.users[t].name
                    }
                    newGroups[i].users.push({
                        id: userIds[j],
                        name: name
                    });
                }
            }

            if (newGroups.length < $scope.groups.length) {
                for (var o=0; o<$scope.groups.length; o++) {
                    var doesExist = false;
```

```
        for (var n=0; n<newGroups.length; n++) {
            doesExist = doesExist || (newGroups[n].id ==
                $scope.groups[o].id);
        }
        if (!doesExist) {
            newGroups.push({
                id: $scope.groups[o].id,
                name: $scope.groups[o].name,
                users: []
            });
        }
    }
    angular.copy(newGroups, $scope.groups);
});
}
});
```

The following two lines basically assign the *reference* pointing to the *users* and *groups* object from the *MyData* factory:

```
$scope.users = MyData.users;
$scope.groups = MyData.groups;
```

This means that if you update the data, you will always have the same data in the controller. There is no need to keep using *get()*.

16. To test the app, you must run it within a simulator or physical device.

## How it works...

For SQLite to initialize properly, the initialization code must start first by checking for the `$ionicPlatform.ready` event before opening the database. This is done to avoid calling the SQLite functions when the device is still loading. If you don't want the app to start afresh with new tables each time the app runs, you can remove all the `DROP TABLE` queries.

To open the database for operations, you can use the following simple `openDB()` function:

```
db = $cordovaSQLite.openDB("my.db", 1);
```

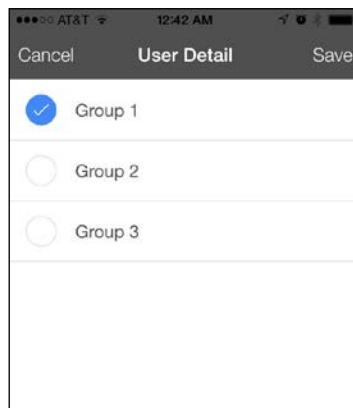
This example leverages the `nestedExecute()` function to run one query after another. The `$cordovaSQLite` only gives you the ability to nest one level. For additional nesting, you have to call `$cordovaSQLite.execute()` again from within the **promise**. Unfortunately, SQLite does not give you the inserted data. So, you have to run the `SELECT` query after successful inserts in order to retrieve all the IDs for each row.

The MyData factory has several key functions to work with SQLite:

- ▶ `addUser()`: This is used to add a user. The `addUser()` function will handle data insertions.
- ▶ `getGroupsByUserId()`: When you click on a specific user, it will pop up the modal and call the `getGroupsByUserId()` function to get a list of all the groups that the user is a member of.
- ▶ `updateGroupByUserId()`: When you change the mapping relationship in the modal, `updateGroupByUserId()` will save all the changes. This update function actually deletes all the existing relationships and inserts new relationships back in the `usergroup` table. In a real-world scenario, you can optimize this process to just `REPLACE`, `INSERT`, or `DELETE`, based on the new data.
- ▶ `getGroupsAll()`: Under the **Groups** menu, you need to call `getGroupsAll()` in order to get all the groups and the members in each group. This `getGroupsAll()` is probably the most complex query in this example, because it performs a `JOIN` operation between the groups and the `usergroup` table. This is required, since you want to get the IDs as well as the names for all the groups. Also, all the user IDs will be aggregated into a string with a comma delimited using `GROUP_CONCAT`. In your controller, you can separate the user IDs again by using the `array_split()` function.

The complexity within `MyData.getGroupsAll()` is due to the fact that you need to translate data from the query into objects. That's why you need to loop through all the rows to get the group information. Then, within each group, you get the list of `userIds` and names. Finally, if the group has no user, you need to also show it by adding back into the array.

The `UsersCtrl` controller is where you tie everything together. If you remember the input box at the end of the list, it is supposed to call `$scope.addUser()`, which will pass the name string to `MyData.addUser()`. When you click on a user, it will open a modal with `$scope.openUserModal()` and then get the group membership from `MyData.getGroupsByUserId(user.id)`. The saving action is done via `MyData.updateGroupByUserId()`:



This controller assumes that the user list (`$scope.users`) and the group list (`$scope.groups`) objects are already available. Actually, these two variables are inherited from the parent `MainCtrl` controller.

Looking under the hood, the SQLite database is just a local file in the device. This means that you may need to think ahead as regards whether it should be backed up or the user is allowed to download. You need to change the second parameter, which is the location option in the following line:

```
db = $cordovaSQLite.openDB("my.db", 1);
```

However, the location option is only applicable for iOS. The following is the description for each option:

Value	Description
0 (default)	Documents that are visible to iTunes and backed up by iCloud
1	The library that is backed up by iCloud and is not visible to iTunes
2	Library/Local Database that is not visible to iTunes and not backed up by iCloud

### There's more...

Other than this, SQLite behaves in a way that is very similar to that of standard SQL. The trade-off for long-term performance and storage capacity is the embedded complexity of working with the SQL queries. In addition to this, the design of SQL intends to eliminate the loss of data that may be caused due to an app crash or a power failure of the device. If you want to completely erase all the SQLite data of your app, use the `deleteDB()` function, as follows:

```
$cordovaSQLite.deleteDB("my.db");
```

### See also

- ▶ You can find out more information about SQLite at <https://www.sqlite.org/>.
- ▶ Currently, the SQLite plugin does not work well with the Windows Phone. You can check out the notes in their documentation by simply navigating to <https://github.com/litehelpers/Cordova-sqlite-storage>.

# 5

## Handling Gestures and Events

In this chapter, we will cover the following tasks related to handling gestures and events:

- ▶ Detecting drag events with a gesture coordinate
- ▶ Communication between a view, controller, and directive using events

### Introduction

It's possible to write a simple app with a handful of pages. However, when the app grows, managing different views and their custom data at a specific time or a triggered event can be very complex. Ionic comes with UI-Router by default. So, you should leverage this advanced routing management mechanism. In general, the following holds true:

- ▶ A view should have its own state, which is basically a JSON object
- ▶ A route (URL) will point to a view and its assigned controller
- ▶ State and view should allow nested views so that you can manage the hierarchy

Since Ionic introduces many new components, you have to understand how those components impact your app's state hierarchy when each state is triggered.

## Detecting drag events with a gesture coordinate

Most mobile UI components leverage dragging to create a better touch-and-feel experience. One common component that you will see everywhere is the scroll list. Another example is the left or right menu. That's why it's important to understand the basic mechanism of a gesture for drag events. It's even possible for you to create custom mobile components using gesture events and pure HTML/CSS/JavaScript.

In this recipe, you will build an app example to understand how to use the data from gesture events. The app will capture drag start, duration, and end events using `$ionicGesture` to give you granular coordinate data for real-time processing. In terms of the UI, you will create a `div` box to allow the dragging and showing of a coordinate.

### Getting ready

Gestures can work on both web and physical devices. However, it's highly recommended to test gestures on a physical device in order to experiment with the screen size and potential performance impacts.

### How to do it...

Here are the instructions to detect drag events with a gesture coordinate:

1. Create a new app using the blank template and go into the folder:  

```
$ ionic start Gesture blank  
$ cd Gesture
```
2. You need to set up the Sass dependencies to style the box, as follows:  

```
$ ionic setup sass
```
3. Open the `index.html` file and replace the `<body>` tag with the following code:  

```
<body ng-app="starter" ng-controller="MainCtrl">  
  <div class="draggable" drammable="pos">  
    <p>{{ pos.x }}, {{ pos.y }}</p>  
  </div>  
</body>
```

You will write more code for MainCtrl later in app.js. The main focus here is actually the `draggable` attribute of the `div` element. This will basically allow you to bind the drag events.

4. Open `app.js` to edit it with the following code:

```
var app = angular.module('starter', ['ionic'])

app.controller('MainCtrl', function($scope) {
  $scope.pos = {
    x: 0,
    y: 0
  });
});
```

This code is used to simply initialize the `pos` object so that you can track the `x` and `y` coordinates later.

5. Now create the `draggable` directive, as follows:

```
app.directive('draggable', function($ionicGesture) {
  return {
    link: function (scope, element, attrs) {
      var elementSize = 100;

      var x = Math.round((window.screen.width -
        elementSize) / 2, 0),
          y = Math.round((window.screen.height -
        elementSize) / 2, 0);

      scope.pos.x = x;
      scope.pos.y = y;

      element[0].style[ionic.CSS.TRANSFORM] =
        'translate3d(' + x + 'px, ' + y + 'px, 0)';

      $ionicGesture.on('dragstart', function(ev) {
        console.log('dragstart: ');
        console.log(ev);
      }, element);

      $ionicGesture.on('dragend', function(ev) {
        console.log('dragend: ');
        console.log(ev);
      });
    }
  };
});
```

```
        x += ev.gesture.deltaX;
        y += ev.gesture.deltaY;
    }, element);

$ionicGesture.on('drag transform', function(ev) {
    console.log('drag transform: ');
    console.log(ev);

    scope.pos.x = Math.round(x + ev.gesture.deltaX, 0);
    scope.pos.y = Math.round(y + ev.gesture.deltaY, 0);
    scope.$digest();

    element[0].style[ionic.CSS.TRANSFORM] =
        'translate3d(' + (x + ev.gesture.deltaX) + 'px, ' +
        (y + ev.gesture.deltaY) + 'px, 0)';
}, element);

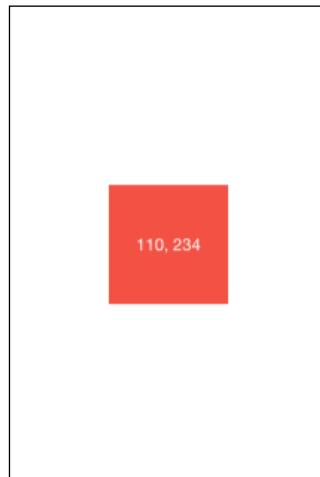
}
}
});
```

This directive will effectively apply to only the `div` element where it was assigned. The purpose is to detect the drag events and appropriately process the user's touch coordinate in order to move the box.

6. To style your app, add the `draggable` class in `/scss/ionic.app.scss` so that it looks like a red box with some coordinate text at the center, as follows:

```
.draggable {
    background-color: $assertive;
    color: white;
    position: absolute;
    width: 100px;
    height: 100px;
    line-height: 100px;
    vertical-align: middle;
    text-align: center;
    z-index: 20;
}
```

7. Test the app in the browser or a physical device. This is the screenshot of the app:



## How it works...

The view and stylesheet for this app is very simple, as this is not the focus in this recipe. Let's take a look at the `draggable` directive in detail. The assumption is that this square box has a size of 100 px:

```
var elementSize = 100;
```

This must also match the `draggable` class in the `scss` file.

The core enabler that is used to detect the drag events is the `$ionicGesture.on()` function. There are three events to watch:

- ▶ `dragstart`: When the drag begins
- ▶ `drag`: During the dragging process
- ▶ `dragend`: When the drag ends

If you run this app in the browser, you can monitor the console for the outputs of these events:

```
dragstart:  
▶ Event {gesture: Object}  
drag:  
▶ Event {gesture: Object}  
dragend:  
▶ Event {gesture: Object}
```

The dragstart and dragend events are only triggered once, while the drag event is continuous. It's very granular. Hence, it gives you a very smooth movement. If you expand the gesture object, which is returned from the drag event, you can see the detailed data:

```
drag:  
▶ Event {gesture: Object} ⓘ  
  bubbles: true  
  cancelBubble: false  
  cancelable: true  
  currentTarget: null  
  defaultPrevented: false  
  eventPhase: 0  
  ▶ gesture: Object  
    angle: 87.45519562018707  
    ▶ center: Object  
      deltaTime: 250  
      deltaX: 0.05599386379279281  
      deltaY: 1.2598619353379092  
      direction: "down"  
      distance: 1.2611056295552043  
      eventType: "move"  
      pointerType: "touch"  
    ▶ preventDefault: function ()  
      rotation: 0  
      scale: 1  
    ▶ srcEvent: TouchEvent  
    ▶ startEvent: Object  
    ▶ stopDetect: function ()  
    ▶ stopPropagation: function ()  
    ▶ target: p.ng-binding  
      timeStamp: 1439260009223  
    ▶ touches: TouchList  
      velocityX: 0.00022397545517117124  
      velocityY: 0.005039447741351637  
    ▶ __proto__: Object  
  ▶ path: Array[6]  
    returnValue: true  
  ▶ srcElement: p.ng-binding  
  ▶ target: p.ng-binding  
  timeStamp: 1439260009225  
  type: "drag"  
  ▶ __proto__: Event
```

There are many types of data available for you to use such as target element, angle, timestamp, direction, and so on. However, your main interest here is to use the `deltaX` and `deltaY` to detect the new position of the touch pointer. The delta values are a measure of how far it is from the original position when the dragging started.

To use the delta values, you just need to change the `pos.x` and `pos.y` values of the scope by adding the delta value for the x and y coordinate, as follows:

```
scope.pos.x = Math.round(x + ev.gesture.deltaX, 0);  
scope.pos.y = Math.round(y + ev.gesture.deltaY, 0);  
scope.$digest();
```

Keep in mind that you must call the `$digest()` function for the AngularJS digest cycle to kick in. The reason behind this is that `$ionicGesture.on()` will not do this for us, as it's outside the default AngularJS controller, directive, or service.

The final part is to apply the new coordinate to the box by changing its transform value, as follows:

```
element[0].style[ionic.CSS.TRANSFORM] = 'translate3d(' + (x +  
ev.gesture.deltaX) + 'px, ' + (y + ev.gesture.deltaY) + 'px, 0)';
```

There obviously are many ways to execute the actual animation. However, using a CSS Transform always gives a better performance and frame rate. Again, `element[0]` is the `div` object that you applied this directive on.

When the drag is performed, you just need to save the new position, as follows:

```
x += ev.gesture.deltaX;  
y += ev.gesture.deltaY;
```

## See also

It's possible to implement velocity with the help of `velocityX` and `velocityY` when the drag event ends. This will allow your object to continue the motion after the touch has completed.

- ▶ Ionic has some default directives to handle gestures such as `on-drag-left`, `on-drag-right`, and so on, which can be found at <http://ionicframework.com/docs/api/directive/onHold/>.
- ▶ However, you may not get the flexibility and rich set of data from `$ionicGesture`: [http://ionicframework.com/docs/api/service/\\$ionicGesture/](http://ionicframework.com/docs/api/service/$ionicGesture/). This is because the default directives hid away the granularity of `$ionicGesture`. For example, you cannot access coordinate data.

## Communication between a view, controller, and directive using events

One of the most confusing aspects of using Ionic is when and how to leverage eventing for communication between various components. Let's take a look at the various scenarios and different ways to handle the data flow:

- ▶ **State to view and/or controller:** This is handled by UI-Router. There are several options. You can use the `resolve` object declared at the route level and pass it to the controller just as a factory. Another option is to pass data as a parameter in `$stateParams` so that the controller for that specific state can receive data.
- ▶ **View to view:** To share data in different views, store them in a parent scope, `$stateParams`, or factory. If it's just a simple piece of data, mainly for display purposes, it's OK to use the parent scope method. However, if the data is specific to state, it should be a part of the state parameters. Factory is always a good practice to keep the shared data between state, view, and controller. You just have to write the getter and setter methods for the factory.
- ▶ **View to controller:** Communication is done via the two-way binding of the `$scope` variables.
- ▶ **Controller to controller:** The data can be passed from the parent to child `$scope`, sent via some type of `$broadcast` event or detected from model change using `$watch`. The example in this section will go over these mechanisms.
- ▶ **Controller to directive:** There are two options here. You can watch the model change from within the directive. This gives you the ability to get an old and a new value. Alternatively, you can pass the model directly to the directive without creating an isolated scope. This directive will use this model with the same reference to the scope model.
- ▶ **Factory to factory:** For internal communication between factories and services, you just need to declare the factory in the function parameters just as a controller.
- ▶ **Directive to factory:** There is normally no need to communicate between a directive and factory. The reason is that your controller should take care as a middle man in this process.

The app example in this recipe will explain how to broadcast an event and watch for a variable change in a controller and directive. In addition, you will have some fun exploring the two types of horizontal scrolling, which are commonly used in many mobile apps.

The following is the screenshot of the app that you will build:



## Getting ready

Since AngularJS and UI-Router comes with the Ionic bundle, there is no need to prepare this in a physical device.

## How to do it...

Here are the instructions to enable communication among a view, controller, and directive using events:

1. Create a new app using the blank template and go into the respective folder, as follows:

```
$ ionic start HorizontalSlide blank  
$ cd HorizontalSlide
```

2. You need to set up the Sass dependencies because you will create a horizontal scroll class later:

```
$ ionic setup sass
```

3. Open the `index.html` file and replace the `<body>` tag with the following:

```
<body ng-app="starter" ng-controller="MainCtrl">
  <ion-pane>
    <ion-header-bar class="bar-stable">
      <h1 class="title">Events</h1>
    </ion-header-bar>
    <ion-content scroll="false">

      </ion-content>
    </ion-pane>
  </body>
```

This is just a skeleton of the app view. You will fill out three more sections within `<ion-content>` later.

4. The first section of the view is for the free-style horizontal scroller:

```
<div class="item item-divider">
  Free Scroll
  <span class="item-note">
    More
  </span>
</div>

<ion-scroll direction="x" class="horizontal-scroll"
padding" ng-controller="FirstCtrl">
  <button class="button button-positive button-large"
ng-click="broadcast('Ionic')">
    <i class="icon ion-ionic"></i>
  </button>
  <button class="button button-calm button-large"
ng-click="broadcast('Twitter')">
    <i class="icon ion-social-twitter"></i>
  </button>
  <button class="button button-balanced button-large"
ng-click="broadcast('Facebook')">
    <i class="icon ion-social-facebook"></i>
  </button>
  <button class="button button-energized button-large"
ng-click="broadcast('Googleplus')">
    <i class="icon ion-social-googleplus"></i>
  </button>
  <button class="button button-assertive button-large"
ng-click="broadcast('Dribbble')">
    <i class="icon ion-social-dribbble"></i>
  </button>
```

```
<button class="button button-royal button-large"
ng-click="broadcast('Octocat')">
    <i class="icon ion-social-octocat"></i>
</button>
<button class="button button-dark button-large"
ng-click="broadcast('Instagram')">
    <i class="icon ion-social-instagram"></i>
</button>
</ion-scroll>
```

This `<ion-scroll>` tag will lock the scroll in the `x` direction only. Otherwise, it behaves just like a regular vertical scroll.

5. The second section is for the slider-style scroll:

```
<div class="item item-divider">
    Slide-by-Slide Scroll
    <span class="item-note">
        More
    </span>
</div>

<div ng-controller="SecondCtrl">
    <ion-slide-box show-pager="false">
        <ion-slide class="padding">
            <button class="button button-block
button-positive" ng-click="changeMyData('POSITIVE')">
                POSITIVE
            </button>
        </ion-slide>
        <ion-slide class="padding">
            <button class="button button-block button-calm"
ng-click="changeMyData('CALM')">
                CALM
            </button>
        </ion-slide>
        <ion-slide class="padding">
            <button class="button button-block button-balanced"
ng-click="changeMyData('BALANCED')">
                BALANCED
            </button>
        </ion-slide>
    </ion-slide-box>
</div>
```

The main difference here is that the scrolling here takes place slide-by-slide, and each slide will take the full width of the screen. Once you slide to the left, it will lock or snap to the second screen.

6. The third section is just for us to view the event and variable. You will see a detailed explanation on this later on:

```
<div class="item item-divider">
  Logs
</div>

<div ng-controller="ThirdCtrl">
  <ul class="list">
    <li class="item">
      <b>myEvent </b> {{ display.value }}
    </li>
    <li class="item">
      <b>myData </b> <my-directive data="myData">
      </my-directive>
    </li>
  </ul>
</div>
```

7. Open app.js and insert the following code in it:

```
var app = angular.module('starter', ['ionic']);

app.controller('MainCtrl', function($scope) {
  $scope.myData = {
    value: 'No value yet'
  }

  $scope.display = {
    value: 'Not fired yet'
  }
});
```

MainCtrl is the parent controller, which initialized two objects (myData and display) for usage within its children controllers.

8. Let's write the three controllers that are needed to handle its three sections in the view earlier:

```
app.controller('FirstCtrl', function($scope, $rootScope) {

  $scope.broadcast = function(param) {
```

```
$rootScope.$broadcast('myEvent', param);
}

});

app.controller('SecondCtrl', function($scope, $rootScope) {

$scope.changeMyData = function(param) {
    $scope.myData.value = param;
}

});

app.controller('ThirdCtrl', function($scope, $rootScope) {

$rootScope.$on('myEvent', function(e, val) {
    $scope.display.value = 'Fired with param: ' + val;
});

});
```

Note that the purpose here is to understand the communication between the three controllers. So, this example is very simple, as it only triggers an event or changes some value. The third controller is your log screen, where you can view what is being changed by detecting the event.

9. Now, you can put together a directive just to test the ability to watch a model change from within the directive:

```
app.directive('myDirective', function($rootScope) {
    console.log('Start directive');
    return {
        template: '{{ myData.value }}',
        scope: {myData : '=data'},
        link: function(scope, element, attrs) {

            scope.$watch(attrs.data, function(newVal, oldVal) {
                if (newVal != oldVal) {
                    console.log('myData has changed');
                    console.log(newVal);
                }
            }, true);
        }
    };
});
```

This directive does not show up anywhere in the view, as it is just for you to understand the mechanism behind the scene.

10. Finally, add the scroll class to the first horizontal scroll in /scss/ionic.app.scss, as follows:

```
.horizontal-scroll {  
    white-space: nowrap;  
    overflow: scroll;  
}
```

11. That's it. Now, you can run the app in the browser to test it.

## How it works...

If you click on any button in the first scroll section, you will see that the log will show the new myEvent object, as shown in the following screenshot:



This is because the first controller has `broadcast()` that triggers `$rootScope.$broadcast()`. You can pass a parameter to this broadcast mechanism as a way to share data. Then, in the third controller, you will see the usage of `$rootScope.$on('myEvent', function(e, val) {})`, as this will listen for `myEvent` specifically, and pass the `val` variable as the parameter value.

There are three things that you should make a note of here:

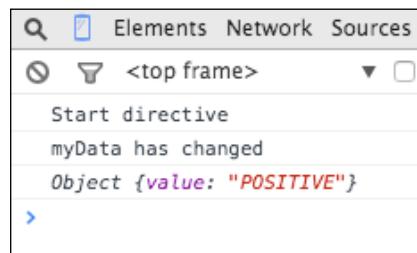
- ▶ You can give the event any name you like. It doesn't matter.
- ▶ You can pass more than one parameter. If you have two or more parameters, just go ahead and add more in the `$on()` function, such as `function(e, val1, val2,...)`.
- ▶ This must be done on `$rootScope`. Basically, firing an event will stick to a specific scope. If you fire an event at the current `$scope` object, the external `$scope` object won't be able to listen to or access the event.

In summary, you use `$broadcast` to broadcast an event, and listen by using `$on` at the `$rootScope` level. Broadcasting is a very useful and easy-to-use mechanism for the communication and sharing of data. However, you should not use broadcast everywhere. The reason behind this is that when your app scales, it's very hard to keep track of and debug all the events being broadcast. Any controller can broadcast and create a new event. You might end up creating duplicate events, or listening to multiple events for the same purpose.

Now, let's evaluate the second scenario using `$watch`. As you know, the AngularJS digest cycle is very important if you want to keep track of variable changes. So, when a controller function changes a variable, it will trigger `$watch` on that variable to execute a function. That's why you are able to see the console output of this code when you click on the **POSITIVE** button:

```
scope.$watch(attrs.data, function(newVal, oldVal) {  
    if (newVal != oldVal) {  
        console.log('myData has changed');  
        console.log(newVal);  
    }  
}, true);
```

This is the output that will show up:



Note that the third parameter is `true`, which means that `$watch` will perform a deep comparison of the object. Otherwise, it will only watch a variable by default and not each key/value inside the object.

In the `myDirective` directive, there is a template with the following string:

```
{{ myData.value }}
```

This tells the view to parse the value here whenever there is a change. This `myData` object is basically brought in by the directive via the following command:

```
scope: {myData : '=data'}
```

Furthermore, if you look at the view declaration, you put an attribute called `data` to reference to the `myData` object of the controller:

```
<my-directive data="myData"></my-directive>
```

The result is the following screen update:

Logs
<b>myEvent</b> Fired with param: Ionic
<b>myData</b> POSITIVE

Now, myData shows POSITIVE as the string in its value key.

Although there are several steps being described here, the concept is that in order to share data from a controller to a directive, you can pass it via the attribute of that directive. Then, you bring it in via the scope declaration of the directive. The rest of the step is similar to how you handle the scope data.

## See also

The best official documentation about \$broadcast and \$watch is the AngularJS website itself. To have a look at it, visit [https://docs.angularjs.org/api/ng/type/\\$rootScope.Scope](https://docs.angularjs.org/api/ng/type/$rootScope.Scope).

You should try to understand more about the detailed mechanism behind \$rootScope in this case.

# 6

## App Theme Customization

In this chapter, we will cover the following tasks related to theme customization:

- ▶ Customizing themes for specific platforms
- ▶ Creating an introduction screen with a custom header

### Introduction

Although Ionic has its own default out-of-the-box themes, you might want to customize your app's look and feel further. There are several methods that can be used to accomplish this. The following are two of these:

- ▶ Change the stylesheet within the Sass file
- ▶ Detect a platform specific to JavaScript and apply custom classes or AngularJS conditions

Either of the aforementioned methods should work, but it's highly recommended that you apply customizations in the Sass file before the app is built in order to achieve maximum rendering performance.

In this chapter, you will create the following three sample authentication apps:

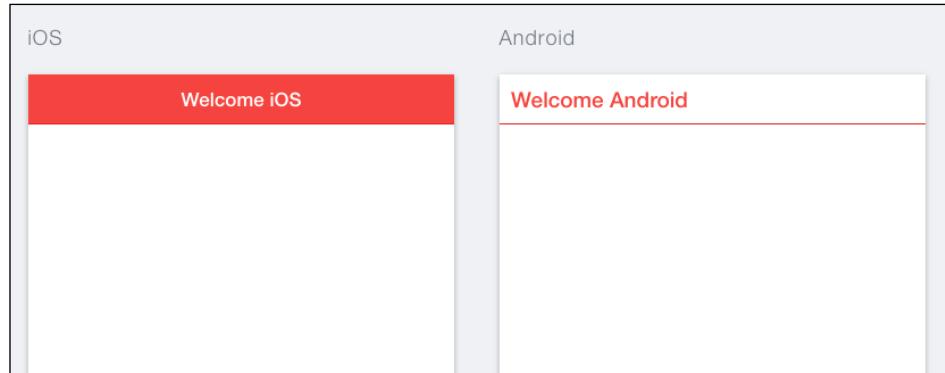
- ▶ Email and password authentication using Firebase
- ▶ Social authentication (Facebook, Twitter, and Google+) using Firebase
- ▶ Social authentication (LinkedIn) using Auth0 and Firebase

Depending on the app, you may not need to use all of these authentication methods. For example, for an app focusing on working professionals, it would make more sense to use a LinkedIn authentication to narrow down the audiences who fit the user profile of the app.

## Customizing themes for specific platforms

Each mobile platform vendor has its own design guideline. This recipe will go over an example of how to address the app theme differently for iOS and Android. In traditional development, using either a native language or other hybrid app solutions, you have to keep separate repositories for each platform in order to customize the theme. This can be very inefficient in the long run.

Ionic makes it very convenient by separating the stylesheet for each platform and specific OS version within the same platform. The example in this recipe covers two possibilities of customization using Sass and JavaScript. The following screenshot shows both the iOS and Android app with different title bar colors and text:



### Getting ready

There is no need to test the theme on a physical device because Ionic can render both iOS and Android in the browser.

### How to do it...

Here are the instructions to customize themes for specific platforms:

1. Create a new app using the blank template and go into the respective folder:

```
$ ionic start Theme blank  
$ cd Theme
```

2. You need to set up the Sass dependencies in the following way, because Ionic uses a number of external libraries for this:

```
$ ionic setup sass
```

3. Open the index.html file and replace the <body> tag with the following:

```
<body ng-app="starter" ng-controller="MainCtrl">
  <ion-pane>
    <ion-header-bar class="bar-assertive">
      <h1 class="title" ng-if="isIOS">Welcome iOS</h1>
      <h1 class="title title-left"
          ng-if="isAndroid">Welcome Android</h1>
    </ion-header-bar>
    <ion-content>
    </ion-content>
  </ion-pane>
</body>
```

You will create the MainCtrl controller later in app.js to handle platform-specific variables (such as isIOS and isAndroid).

4. Open app.js and add the controller, as follows:

```
.controller('MainCtrl', function($scope) {
  $scope.isIOS = ionic.Platform.isIOS();
  $scope.isAndroid = ionic.Platform.isAndroid();
});
```

5. Open the ionic.app.scss file under ./scss and add the following code at the end after @import "www/lib/ionic/scss/ionic":

```
.platform-android {
  .bar {
    background-color: white !important;
  }

  .bar .title {
    color: $assertive !important;
  }
}
```

This simply overrides the Android theme with a customized bar and title classes.



This .scss file is in the scss folder in your root project. You don't want to directly modify the default .scss files of Ionic under ./www/lib/ionic/scss, because it's not a good practice.

6. Conduct a test run of the app in the browser, as follows:

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

7. If you inspect the DOM, you will see that Android-specific classes are injected into the DOM:



## How it works...

Ionic automatically created platform-specific parent classes and put them at the `<body>` tag. The iOS app will include the `.platform-ios` class. The Android app will have `.platform-android4_4` for Android v4.4, `.platform-android5_0` for Android v5.0, or just `.platform-android`. So, for stylesheet customization, you can leverage these existing classes to change the look and feel of your app.

The example also leverages an existing function in an `ionic` object to detect platforms:

```
$scope.isIOS = ionic.Platform.isIOS();  
$scope.isAndroid = ionic.Platform.isAndroid();
```

By making the scope variables available to the view, you can use it to hide or show a specific DOM using `ng-if`. It's recommended that you use `ng-if` instead of `ng-show` because `ng-show` may show and hide the element right away, thus creating a *flickering* effect.

## There's more...

It would be better if you could take a look at several `.scss` files under `./www/lib/ionic/scss` to understand how Ionic builds its own `.css` files. Each component has its own `.scss` file. The `_variables.scss` file will give you all the global theme variables that you can override in `./scss/ionic.app.scss`.

## See also

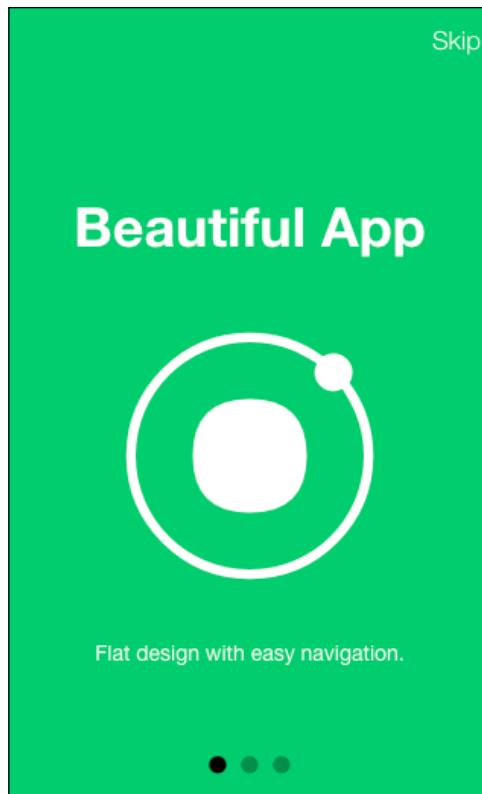
- ▶ There is other device information available from the `ionic.Platform` object. You can even detect iPad devices. For more information, visit <http://ionicframework.com/docs/api/utility/ionic.Platform/>.

- ▶ The Ionic website also has some basic instructions on how to write a Sass theme. For more information on this, visit <http://learn.ionicframework.com/formulas/working-with-sass/>.

## Creating an introduction screen with a custom header

Every app has a different onboarding experience. Some will show a short video, while others walk the users through a few introduction slides. Creating slides is useful because you can pitch initial key messages to set an expectation for users. Also, they can move between slides at their own pace. Most apps will have three to five slides for an introduction screen.

In this recipe, you will learn how to create a three-slide intro screen using `<ion-slide-box>`, which is an Ionic directive that is used to create horizontal slides. In addition to this, you will incorporate ngStorage to detect whether the user has seen the introduction or not. This is useful when you don't want users to keep seeing the introduction every time the app starts. Here is the screenshot of the app:



## Getting ready

Since ngStorage can work locally in the browser, there is no need to test this app on a physical device.

## How to do it...

Here are the instructions to create an introduction screen with custom header:

1. Create a new app using the blank template and go into the folder:

```
$ ionic start Intro blank  
$ cd Intro
```

2. Set up the Sass dependencies, as follows:

```
$ ionic setup sass
```

3. Install ngStorage by executing the following command:

```
$ bower install ngstorage
```



ngstorage is in lowercase in the command because ngStorage is a different repository.

4. Open index.html and include the ngStorage script:

```
<script src="lib/ngstorage/ngStorage.js"></script>
```

5. For the body content, all you need to do is place a *view holder*, as follows:

```
<body ng-app="starter">  
  <ion-nav-view></ion-nav-view>  
</body>
```

This is basically just used to hold the additional templates of the intro and app screen.

6. First, let's knock off the easy template—app.html in the ./www/templates folder:

```
<ion-pane>  
  <ion-header-bar class="bar-assertive">  
    <h1 class="title">Welcome</h1>  
  </ion-header-bar>  
  <ion-content class="has-header">  
    Hello World!  
  </ion-content>  
</ion-pane>
```

This template is just the app screen that you want the user to go to after viewing the introduction. Since this is not the focus of this example, you can make it super simple.

7. Create `intro.html` in the `templates` folder under `./www`, as follows:

```
<ion-nav-view class="intro">
  <ion-header-bar align-title="middle" class="bar-balanced
  bar">
    <button class="button button-icon icon
    ion-ios-arrow-back" ng-if="slides.currentSlide > 0"
    ng-click="slides.currentSlide = slides.currentSlide-1">
    </button>
    <span class="title" ng-bind-html="title"></span>
    <button class="button button-clear" ui-sref="app">
      Skip
    </button>
  </ion-header-bar>
  <ion-content class="has-footer" scroll="false">
    <ion-slide-box active-slide="slides.currentSlide"
    on-slide-changed="slideChanged(index)">
      <ion-slide class="center">
        <div class="row">
          <div class="col">
            <h1>Beautiful App</h1>
            <div class="center-image">
              <i class="icon ion-ionic"></i>
            </div>
            <p>Flat design with easy navigation.</p>
          </div>
        </div>
      </ion-slide>
      <ion-slide class="center">
        <div class="row">
          <div class="col">
            <h1>Useful Content</h1>
            <div class="center-image">
              <i class="icon ion-ios-paper"></i>
            </div>
            <p>Daily update content for your needs.</p>
          </div>
        </div>
      </ion-slide>
      <ion-slide class="center">
        <div class="row">
```

```
<div class="col">
  <h1>Lowest Price</h1>
  <div class="center-image">
    <i class="icon ion-social-usd"></i>
  </div>
  <p>Beat all competitor offerings.</p>
</div>
</div>
</ion-slide>
</ion-slide-box>
</ion-content>
<ion-footer-bar align-title="center" class="bar-balanced
darker-balanced" ui-sref="app" ng-if="slides.currentSlide
== 2">
  <span class="title">Get Started</span>
</ion-footer-bar>
</ion-nav-view>
```

Your intro screen is actually very simple. It has a header bar and content area. Within the content area, you will use `<ion-slide-box>` to display the sliders.

8. To create the logic as regards whether to show the intro screen, you have to write this component when the app runs. Edit `app.js` in the following way:

```
var app = angular.module('starter', ['ionic',
'ngStorage']);

app.run(function($ionicPlatform, $rootScope, $state,
$localStorage) {
  $ionicPlatform.ready(function() {
    $rootScope.$storage = $localStorage.$default({
      seenIntro: false
    });

    if ($rootScope.$storage.seenIntro) {
      event.preventDefault();
      $state.go('app');
    }
  });
});
```

ngStorage is the key here because it allows you to detect \$rootScope.\$storage.seenIntro even when the app restarts.

9. Next, you need to set up the routing for the app, as follows:

```
.config(function($stateProvider, $urlRouterProvider) {  
  
    $stateProvider  
  
        .state('intro', {  
            url: "/",  
            templateUrl: "templates/intro.html",  
            controller: 'IntroCtrl'  
        })  
  
        .state('app', {  
            url: "/app",  
            templateUrl: "templates/app.html"  
        });  
  
    $urlRouterProvider.otherwise('/');  
})
```

As mentioned earlier, intro.html is your intro template with route /, while app.html is the main app route /app.

10. There is no need to handle anything for the app controller, but you need to do a custom title and create a slideChanged() function, as follows:

```
.controller('IntroCtrl', function($scope, $rootScope) {  
    $scope.slides = {  
        currentSlide: 0  
    };  
    $scope.title = '<i class="icon ion-android-home"></i>';  
  
    $scope.slideChanged = function(index) {  
        $scope.slideIndex = index;  
        if (index == 2)  
            $rootScope.$storage.seenIntro = true;  
    };  
});
```



When the user is on the last slide (when the index is 2), you set `$rootScope.$storage.seenIntro` to true, which indicates that the user has seen the entire intro screen.

11. Now, the critical part is to update the `ionic.app.scss` file under `./scss` to apply proper styling for the introductory screen. Let's start with the button in the last screen:

```
$balanced-darker: darken($balanced, 10%) !default;
```

```
.darker-balanced {  
  background: $balanced-darker !important;  
}
```

Basically, `$balanced-darker` is a darker color, which is created using the `$balanced` variable in Ionic.

12. Since the entire `intro` template has a `.intro` class, you can style all the child elements independently, as follows:

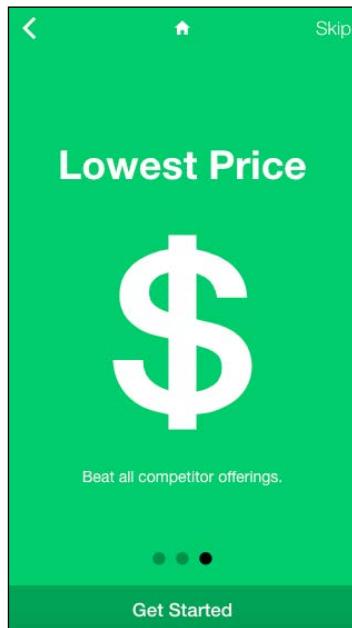
```
.intro, .intro .scroll-content {  
  background-color: $balanced!important;  
}  
  
.intro .scroll, .intro .slider {  
  height: 100%!important;  
  background-color: $balanced;  
}  
  
.intro .center {  
  display: -webkit-box;  
  display: -moz-box;  
  display: -ms-flexbox;  
  display: -webkit-flex;  
  display: flex;  
  -webkit-box-direction: normal;  
  -moz-box-direction: normal;  
  -webkit-box-orient: horizontal;  
  -moz-box-orient: horizontal;  
  -webkit-flex-direction: row;  
  -ms-flex-direction: row;  
  flex-direction: row;
```

```
-webkit-flex-wrap: nowrap;  
-ms-flex-wrap: nowrap;  
flex-wrap: nowrap;  
-webkit-box-pack: center;  
-moz-box-pack: center;  
-webkit-justify-content: center;  
-ms-flex-pack: center;  
justify-content: center;  
-webkit-align-content: stretch;  
-ms-flex-line-pack: stretch;  
align-content: stretch;  
-webkit-box-align: center;  
-moz-box-align: center;  
-webkit-align-items: center;  
-ms-flex-align: center;  
align-items: center;  
}  
  
.intro .center-image {  
    text-align:center;  
}  
  
.intro .center-image i {  
    font-size: 200px;  
    color: white;  
}  
  
.intro h1, .intro p {  
    color: white;  
    text-align: center;  
    padding: 20px 0 20px 0;  
}  
  
.intro h1 {  
    font-weight: bold;  
}
```

13. It's time to test the app in the browser:

```
$ ionic serve
```

14. If you go to the last screen, the button will show up. In addition to this, the back button to the top left will only be shown when there is a slide before the current one that you can go to:



## How it works...

This app has several *moving parts* that you have to connect together:

- ▶ A Local Storage's `seenIntro` variable will be checked each time the app starts. Since `run()` will be executed first, you have to place the check logic there. The `$localStorage.$default()` function is used to set the default value if the variable doesn't exist yet.
- ▶ The `view-title` does not take HTML code. So, you have to place a scope variable at `<span class="title" ng-bind-html="title"></span>` to update.
- ▶ It's important to keep track of the current slide when the slide is changed by using `active-slide="slides.currentSlide"`. Once the slide is changed it will trigger an event, which can be assigned to a directive called **on-slide-changed**.
- ▶ In order to show or hide the **Back** button, you just need to check whether the current slide index is greater than zero by using `ng-if="slides.currentSlide > 0"`.
- ▶ In your footer, it's basically a button that can be used to go to the app state and only show whether the current slide index is two: `<ion-footer-bar align="center" class="bar-balanced darker-balanced" ui-sref="app" ng-if="slides.currentSlide == 2">`.

It's good practice to not change Ionic classes directly, but create a parent class for the page. In this case, you created an `intro` class so that you can modify everything in that page specifically. The Ionic slider does not *stretch* vertically by default. That's why you need to apply its height as 100%.

All the other elements within a specific slide can be customized as you wish. Normally, it's good enough to keep things simple, with a header text as your value proposition and an image and a short one-liner to explain the idea.



# 7

## Extending Ionic with Your Own Components

In this chapter, we will cover the following tasks related to creating custom directives and filters:

- ▶ Creating a scroll progress bar directive
- ▶ Creating a custom filter
- ▶ Animating an app using `requestAnimationFrame` with event binding

### Introduction

Although Ionic has many out-of-the-box components, you may want to start customizing the existing components or create your own. There are many scenarios where you just cannot fit the default components in your specific use cases. Here are some of the examples of the scenarios:

- ▶ Custom input fields
- ▶ The formatting of data with your specific requirements
- ▶ The binding of your own UI component with Ionic events
- ▶ The creation of animation based on stateful events

Alternatively, you can reuse the AngularJS Bootstrap module and just change its CSS to create a consistent look and feel. Since you can leverage web technologies, creating new components in Ionic is very simple. It's the same process as building new directives and filters for the AngularJS Single-Page Application. If you are unfamiliar with this, you can start with the following sections and go through some simple examples.

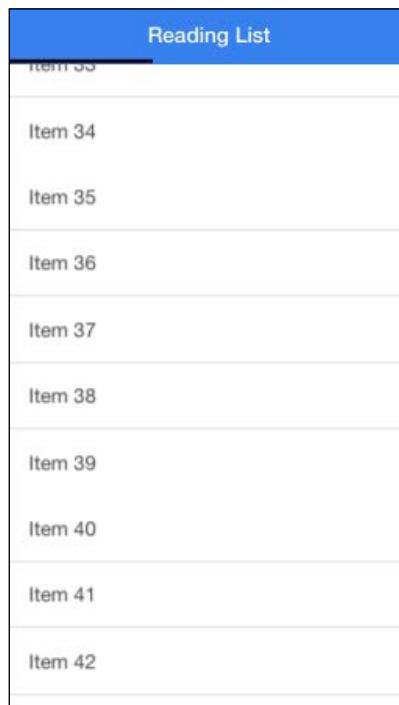
You will learn the following in this chapter:

- ▶ How to create custom directives and filters
- ▶ How to look for Ionic events and bind them to create stateful animation

## Creating a scroll progress bar directive

Progress bars are very useful, as they indicate the amount of task that is completed. It gives users a great experience in terms of what to expect (or how long the process should take to finish). In content websites such as blogs, you may have observed that the progress bar is also used to show how much the users have read. In this recipe, you will create a custom directive to show the reading progress. The concept is very simple. This directive basically checks against the scroll position of `<ion-content>` in order to adjust the bar (`div`) length.

The following is the screenshot of the app:



There are 100 items in the list. As you scroll, the bar at the top will move from left to right to indicate the percentage of reading completed.

## Getting ready

This app example can work either in a browser or physical device.

## How to do it...

Here are the instructions to create a scroll progressive bar directive:

1. Create a new app using a *blank* template and go into the folder:

```
$ ionic start ScrollProgress blank  
$ cd ScrollProgress
```

2. Set up the Sass dependencies for the custom progress bar div, as follows:

```
$ ionic setup sass
```

3. Open the `index.html` file and replace the `<body>` tag with the following:

```
<body ng-app="starter" ng-controller="MyCtrl">  
  
<ion-header-bar class="bar-positive">  
  <h1 class="title">Reading List</h1>  
  <scroll-progress></scroll-progress>  
</ion-header-bar>  
  
<ion-content>  
  <ion-list>  
    <ion-item ng-repeat="item in items">  
      Item {{ item.id }}  
    </ion-item>  
  </ion-list>  
</ion-content>  
  
</body>
```

There is nothing special about the view structure. What you have here is just a header bar and a list of items. However, there is a new `scroll-progress` directive to indicate the scroll progress of the list.

4. Open `app.js` to edit it with the following code:

```
var app = angular.module('starter', ['ionic']);  
  
app.controller('MyCtrl', function($scope, $ionicScrollDelegate) {  
  
  $scope.items = [];
```

```
        for (var i=1; i<=100; i++) {
            $scope.items.push({ id: i });
        }

    });

});
```

Again, to simplify the unimportant part, this `MyCtrl` controller will just populate the list with 100 items.

5. To create the custom scroll progress directive, you need to add the following code:

```
app.directive('scrollProgress',
function($ionicScrollDelegate) {
    return {
        template: '<div class="progress"
style="{{percentage}}\''></div>',
        link: function (scope, element, attrs) {
            scope.percentage = '0%';

            ionic.DomUtil.ready(function() {
                var windowHeight = $ionicScrollDelegate.
                _instances[0].element.clientHeight,
                    scrollHeight = $ionicScrollDelegate.
                    _instances[0].element.querySelector(
                    'div.scroll').clientHeight,
                    delta = scrollHeight - windowHeight;

                $ionicScrollDelegate._instances[0].
                $element.bind('scroll', function(e) {
                    var scrollPosition = $ionicScrollDelegate.
                    getScrollPosition().top;
                    scope.percentage = 'width:
                    ' + (scrollPosition / delta * 100) + '%';
                    scope.$digest();
                });
            });
        }
    };
});
```

This custom directive will use `$ionicScrollDelegate`, which is a standard out-of-the-box delegate from Ionic that is used to get and set the position of the scroller.

- Finally, just add the styling for the `progress` class in `ionic.app.scss`, as follows:

```
.progress {  
  height: 3px;  
  background: darken(#33cd5f, 70%);  
  position: absolute;  
  left: 0;  
  bottom: 0;  
}
```

## How it works...

The progress bar is just a `div` element with the width as a percentage. It must be a part of the header, as it is positioned between the header and the main content.

It's important to understand how Ionic structures its content and scroll element. In order to access the information of these elements, you need to work through `$ionicScrollDelegate`. There are four variables that can be used to calculate the percentage:

```
var windowHeight = $ionicScrollDelegate.  
  _instances[0].element.clientHeight,  
  scrollHeight = $ionicScrollDelegate._instances[0]  
  .element.querySelector('div.scroll').clientHeight,  
  delta = scrollHeight - windowHeight;
```

This also can be used:

```
var scrollPosition =  
$ionicScrollDelegate.getScrollPosition().top;
```

The `windowHeight` variable is the height of the device screen or the visible content area. The `scrollHeight` variable is the real height of the scrollable element (`div.scroll`). So, when you scroll, this element will change its top position to simulate the scrolling effect. The `delta` variable is used to calculate the percentage. By default, you will see the first screen. So, you need to subtract the height of the screen to get your starting point. The `scrollPosition` variable is the top position of `div.scroll`, and this variable is updated every time the binding to the `scroll` event is triggered.

One tricky part that you should be aware of is the difference between `element` and `$element`. Here's the `element` part:

```
$ionicScrollDelegate._instances[0].element
```

Here's the `$element` part:

```
$ionicScrollDelegate._instances[0].$element
```

This is an undocumented area, which requires you to carefully inspect in the console before using. Eventually, any `<ion-content>` directive will create a `$ionicScrollDelegate._instances[0]` instance in its delegate. You can look at the values of this object to find out the difference between `element` and `$element`:

```
▼ Object { _scrollViewOptions: Object, element: ion-content.top-image.scroll-content.ionic-scroll.has-header,
  $delegateHandle: undefined
  ► $filterFn: function ()
  ▼ $element: JQLite[1]
    ► 0: ion-content.top-image.scroll-content.ionic-scroll.has-header
      length: 1
      ► __proto__: Object[0]
    ► _timeout: function timeout(fn, delay, invokeApply)
    ► _scrollViewOptions: Object
    ► _setRefresher: function (refresherScope, refresherElement, refresherMethods)
    ► _anchorScroll: function (shouldAnimate)
  ▼ element: ion-content.top-image.scroll-content.ionic-scroll.has-header
    accessKey: ""
    ► attributes: NamedNodeMap
    baseURI: "http://localhost:8100/"
    childElementCount: 2
    ► childNodes: NodeList[2]
    ► children: HTMLCollection[2]
    ► classList: DOMTokenList[4]
      className: "top-image scroll-content ionic-scroll has-header"
    clientHeight: 526
    clientLeft: 0
    clientTop: 0
    clientWidth: 320
    contentEditable: "inherit"
    ► dataset: DOMStringMap
    dir: ""
    draggable: false
    ► firstChild: div.scroll
    ► firstElementChild: div.scroll
    hidden: false
    id: ""
    innerHTML: "<div class='scroll' style='transform: translate3d(0px, 0px, 0px) scale(1);'><ion-list cl
    innerText: "Item 1~Item 2~Item 3~Item 4~Item 5~Item 6~Item 7~Item 8~Item 9~Item 10~Item 11~Item 12~Item"
```

Basically, the `element` property is the actual DOM element, while `$element` is an AngularJS version (or JQLite) of this. You can use `element` to get the DOM properties such as height and width. However, the `scroll` binding is only available in `$element`, as it is a specific functionality of Ionic.

When you create a custom directive, you can insert your own template within that directive, as follows:

```
template: '<div class="progress" style=\'{\{percentage\}}\}></div>'
```

In this case, the `percentage` variable will be interpolated into the `style` attribute of this `div` element.

Whenever you are interacting with the out-of-the-box elements of Ionic, you need to make sure that they are rendered completely by putting your code within the `ionic.DomUtil.ready` event. In addition to this, the `bind()` function will not trigger the AngularJS scope digest cycle. That's why you need to call `scope.$digest()` manually within the scroll binding. This is the only way AngularJS knows how to look for changes and update the view with your new percentage value.

## See also

For more information about creating a custom directive, check out the AngularJS documentation for directives by visiting <https://docs.angularjs.org/guide/directive>.

## Creating a custom filter

Filters are a feature of AngularJS and not specific to Ionic. The main reason you might want to use a filter is when you just need the data to be displayed in a different format in the view. You don't want to change the actual value in the controller or factory. This makes things very convenient because you don't have to decide upon a specific format within the controller code while leaving the flexibility in the view component.

Here is the list of some out-of-the-box filters (from <https://docs.angularjs.org/api/ng/filter>):

- ▶ currency
- ▶ number
- ▶ date
- ▶ json
- ▶ lowercase
- ▶ uppercase
- ▶ limitTo
- ▶ orderBy

In this recipe, you will learn how to add more filters using the `angular-filter` module as well as create a custom filter. The following is the screenshot of the app:



## Getting ready

There is no need to test in a physical device because the AngularJS filter will work just fine in a web browser.

## How to do it...

Here are the instructions to create a custom filter:

1. Create a new app using the `blank` template and go into the respective folder:

```
$ ionic start Filter blank  
$ cd Filter
```

2. You need to install `angular-filter` so that you can use it in one of the examples:

```
$ bower install angular-filter --save
```

3. Open the `index.html` file and include the dependency, as follows:

```
<script src="lib/angular-filter/dist/  
angular-filter.js"></script>
```

4. Then, replace the `<body>` tag with the following:

```
<body ng-app="starter" ng-controller="MainCtrl">
  <ion-pane>
    <ion-header-bar class="bar-stable">
      <h1 class="title">Ionic Filter</h1>
    </ion-header-bar>
    <ion-content>
      <div class="list">
        <div class="item item-divider">
          Use $filter in controller
        </div>
        <label class="item item-input">
          <input type="text" placeholder="Type anything
lowercase here" ng-model="data.lowercase">
        </label>
        <label class="item">
          {{ data.uppercase }}
        </label>
        <div class="item item-divider">
          Use angular-filter in interpolation
        </div>
        <label class="item">
          Max of [1,2,3,4,7,8,9] = {{ numberArray | max }}
        </label>
        <div class="item item-divider">
          Create custom filter
        </div>
        <label class="item" ng-repeat="item in
countryCodes">
          {{ item }} <i class="icon ion-ios-arrow-
thin-right"></i> {{ item | languageName }}
        </label>
      </div>
    </ion-content>
  </ion-pane>
</body>
```

The app will have three sections:

- ▶ How to tighten the input model in real time with a filter from the controller
- ▶ How to use the angular-filter module
- ▶ How to set up your own custom filter

5. Open `app.js` and edit it with the following code:

```
var app = angular.module('starter', ['ionic',
'angular.filter']);

app.controller('MainCtrl', function($scope, $filter) {
  $scope.data = {
    lowercase: '',
    uppercase: 'Resulted Upper Case'
  }

  $scope.$watch('data.lowercase', function(newVal,
oldVal) {
    if (newVal != oldVal)
      $scope.data.uppercase = $filter('ucfirst')(newVal);
  });

  $scope.numberArray = [1,2,3,4,7,8,9];
  $scope.countryCodes = ["IN", "MX", "US", "GB", "FR"];
}) ;
```

This code is used to set up your controller with some data so that it can be used in the filters.

6. Next, create a custom filter, as follows:

```
app.filter('languageName', function() {
  var codes = {
    "BR": "Brazil",
    "CA": "Canada",
    "CN": "China",
    "FR": "France",
    "DE": "Germany",
    "IN": "India",
    "IL": "Israel",
    "IT": "Italy",
    "MX": "Mexico",
    "US": "United States"
  }

  return function(input) {
    var output = codes[input] ? codes[input] : "Unknown";
    return output;
  }
});
```

This is a very simple, hardcoded example that demonstrates how to create a filter. The goal is to perform a conversion from the `input` value to the `output` value.

7. There is no need to customize any CSS. So, you can run and test the filter in the browser.

## How it works...

You can use the AngularJS filter in the view or as a function in the controller or factory. It's considered a utility that simply converts or transforms any value to a desired value. There is no limitation on how you want to structure the filter.

In the first example of using `$filter` in the controller, you just use `$scope.$watch` on the input model and convert it in real time, as follows:

```
$scope.$watch('data.lowercase', function(newVal, oldVal) {  
    if (newVal != oldVal)  
        $scope.data.uppercase = $filter('ucfirst')(newVal);  
});
```

You must declare `$filter` as a dependency for that controller.

The second example illustrates how a filter is used directly in the view without an interference of controller or factory:

```
{ { numberArray | max } }
```

AngularJS automatically detects the `|` sign and turns the value that is in front of it to an input. The `max` function will use it for conversion.

Finally, your last example is used to add the `languageName` filter. It's basically just a function that returns another function in the following format:

```
return function(input) {  
    var output = codes[input] ? codes[input] : "Unknown";  
    return output;  
}
```

The value of the output will be what AngularJS renders in the view.

## See also

- ▶ To understand more about the AngularJS filter, you can check out the official documentation at <https://docs.angularjs.org/api/ng/filter/filter>.
- ▶ `Angular-filter` has a wide range of filters to choose from. You can get a full list at <https://github.com/a8m/angular-filter>.

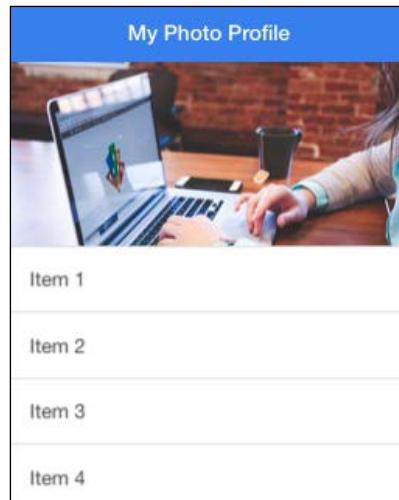
## Animating an app using requestAnimationFrame with event binding

Animation is always a tricky part when it comes to mobile app development. The main reason behind this is that you really have to know how JavaScript or CSS animation works. Otherwise, if you use external libraries, you will run into performance problems such as jerky movements. In addition to this, there are many ways that can be used to bind an animation stage with a native touch event of the device. So, it always takes a little extra effort to get a good result.

In this recipe, you will learn how to:

- ▶ Detect a scroll event
- ▶ Bind the scroll position on pull down in order to slightly zoom the image
- ▶ The image zoom has exact ration is the scroll bounce

This example is similar to many apps with a profile page such as a Twitter iOS app. The top image can be the user's profile picture. The following is a screenshot of the app that you will build:



### Getting ready

You can test this in a web browser. However, it's recommended that you use a physical device to test the animation performance, especially during fast scroll movements.

## How to do it...

Here are the instructions to animate an app using `requestAnimationFrame` with event binding:

1. Create a new app using the `blank` template and go into the respective folder:

```
$ ionic start ZoomOnScroll blank  
$ cd ZoomOnScroll
```

2. You need to set up the Sass dependencies, as follows:

```
$ ionic setup sass
```

3. Open the `index.html` file and replace the `<body>` tag with the following:

```
<body ng-app="starter" ng-controller="MyCtrl">  
  
<ion-header-bar class="bar-positive">  
  <h1 class="title">My Photo Profile</h1>  
</ion-header-bar>  
  
<ion-content class="top-image" zoom-on-scroll>  
  <ion-list>  
    <ion-item ng-repeat="item in items">  
      Item {{ item.id }}  
    </ion-item>  
  </ion-list>  
</ion-content>  
  
</body>
```

There is no image element in this code, because the image is actually a part of the `<ion-content>` background.

4. Open `ionic.app.scss` placed under `scss` and add the following code:

```
.top-image {  
  padding-top: 150px!important;  
}  
  
.scroll-content {  
  background-image: url('../img/sample.jpg');  
  background-size: 100%;  
  background-repeat: no-repeat;  
}
```

This will ensure that the content has 150 px of free space at the top, which will show the sample.jpg image.

5. Open app.js to edit with the following code:

```
var app = angular.module('starter', ['ionic']);

app.controller('MyCtrl', function($scope, $ionicScrollDelegate) {

    $scope.items = [];
    for (var i=1; i<=100; i++) {
        $scope.items.push({ id: i });
    }

});
```

6. To handle the animation interaction, you need to create the zoomOnScroll directive, which will be placed in <ion-content>:

```
app.directive('zoomOnScroll', function($ionicScrollDelegate) {
    return {
        link: function (scope, element, attrs) {
            ionic.DomUtil.ready(function() {

                var scrollContent =
                    document.querySelector('.scroll-content'),
                    windowHeight = $ionicScrollDelegate.
                    _instances[0].element.clientHeight;
                windowWidth = $ionicScrollDelegate._
                    instances[0].element.clientWidth;

                $ionicScrollDelegate.
                _instances[0].$element.bind('scroll', function(e) {

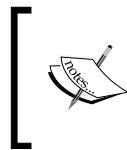
                    function callback() {
                        var scrollPosition = $ionicScrollDelegate.
                        getScrollPosition().top,
                            zoom = (-(scrollPosition*2 /
                            windowHeight) + 1) * 100,
                            offset = (windowWidth * ((zoom -
                            100) / 100)) / 2;

                        if (zoom > 100) {
                            scrollContent.style['background-size']
                            = zoom + '%';
                            scrollContent.style['background-
                            position-x'] = -offset + 'px';
                        }
                    }
                });
            });
        }
    };
});
```

```
        }
    }

    ionic.DomUtil.requestAnimationFrame(callback);

    }) ;
  });
}
);
}) ;
```



While you can use the regular method for animation or even a timeout event, in this case, it's better to use `ionic.DomUtil.requestAnimationFrame()` for improved performance. More on this will be discussed later.

7. Test run the app in the browser, as follows:  
`$ ionic serve`
8. Alternatively, you can perform a test run of the app in an iOS device when the phone is connected via a USB, as follows:  
`$ ionic run -device`
9. When you drag or pull down the list, the image will expand to zoom, as shown in the following screenshot (without distortion):



## How it works...

Before explaining in depth, it's good to know about several limitations of this example. Some of these limitations are as follows:

- ▶ The code assumes that your top image is 150 px in height. This is just an arbitrary number and can be changed.
- ▶ The actual image size is 1280 x 853. So, if you want to swap out with other images, please consider testing the ratio properly (to avoid white space or missing corners).
- ▶ The image URL is fixed in the CSS. To have dynamic images (such as per user profile), you should populate the CSS background-image property properly from the AngularJS code.
- ▶ There is a potential performance impact for low-end Android devices. It's highly recommended that you build using Crosswalk for a better animation performance.

Let's start by taking a look at the `zoomOnScroll` directive. You should only start processing the DOM events when Ionic finishes its rendering:

```
ionic.DomUtil.ready()
```

Otherwise, you may get a strange bug when the binding sometimes happens and sometimes it does not.

There are some undocumented features of Ionic regarding `<ion-content>`. First, when you create a content area, it will insert a `div` element with the `scroll-content` class. That's why you should grab it in order to animate its background CSS properties, as follows:

```
var scrollContent = document.querySelector('.scroll-content')
```

Another area is the use of `$ionicScrollDelegate._instances[0].$element` to create a scroll binding. This scroll binding isn't available if you use a normal DOM element object.

To understand the scroll event and the data that you can get from it, you can output the variables from within the `callback()` function, as follows:

```
console.log(scrollPosition);
console.log(zoom);
console.log(offset);
```

What you will see is something like the following when the list content is pulled or dragged down:

-4.375
101.66349809885931
2.6615969581748917
-10
103.8022813688213
6.083650190114076
-27.5
110.45627376425855
16.730038022813687
-27.5
110.45627376425855
16.730038022813687
-43.75
116.63498098859316
26.61596958174905
-43.75
116.63498098859316
26.61596958174905
-48.75
118.5361216730038
29.657794676806088

The top position actually can be a negative number when it goes beyond the topmost position of the content area:

```
scrollPosition = $ionicScrollDelegate.getScrollPosition().top
```

That's where you take advantage of this feature to calculate the zoom ratio, which is the second line:

```
zoom = (-(scrollPosition*2 / windowHeight) + 1) * 100
```

The zoom value is just a percentage number to zoom the background image using the `background-size` CSS property. You can generate this ratio any way you like using the `scrollPosition` value. In this case, it is two times the amount of scrolling (to avoid seeing a white space area below the image).

However, this is not enough to complete this visual effect. You should also make sure that the zoomed image is at the center all the time. To do this, calculate the offset position, as follows:

```
offset = (windowWidth * ((zoom - 100) / 100)) / 2
```

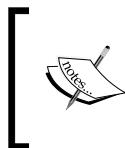
This means that if the image is wider than the device's width, it should reposition the image to the left so that it stays in the middle.

Once all the required values are set, you just have to change the CSS properties of the scrollContent object, as follows:

```
scrollContent.style['background-size'] = zoom + '%';
scrollContent.style['background-position-x'] = -offset + 'px';
```

To improve the animation performance, you must use `requestAnimationFrame`, as follows:

```
ionic.DomUtil.requestAnimationFrame(callback);
```



Ionic has abstracted the `requestAnimationFrame` function based on the native `window.requestAnimationFrame()` function. This feature works nicely with the browser rendering engine. In each second, the browser repaints the view approximately 60 times.



By calling `requestAnimationFrame`, you basically tell the browser to execute the `callback()` function before the next repaint. This is a great way to get into *alignment* with the rendering engine to make sure that you don't miss any frame. You may ask why people can't use the `setInterval()` method to manually inject the rendering code 60 times per second. It's simple to clarify that `setInterval()` does not obey the exact completion of the execution to ensure that it's executed 60 times per second.

There are always some short delays when you use `setInterval()`, and the 60 frames may run for over more than one second. For smooth animation, it's critical to understand this concept.

### There's more...

The difference between the animation performances when using pure CSS transitions versus JavaScript has been an ongoing debate. However, with proper implementation of JavaScript, it has been tested and proven that JavaScript can make faster animation with a higher frame per second. An article, which can be viewed by visiting <http://davidwalsh.name/css-js-animation>, depicts some very useful facts.

Monitoring the frame rate in animation is extremely important. In general, the CSS transition is good for stateless and simple UI animation. JavaScript animation should be used when the animation is triggered by JavaScript and there is a granular transition process between state A and B. The example in this section is a good use of `requestAnimationFrame` because it's hard to predict the exact animation for the scrolling inputs from the device.

This example app is simple enough to help you just animate by changing the background-size and background-position-x properties. There are many high-performance animation engines in the market that can help you perform complex animation scenarios. You can check out Greensock (<http://www.greensock.com>) and Velocity.js (<http://velocityjs.org/>) to find out more.



# 8

## User Registration and Authentication

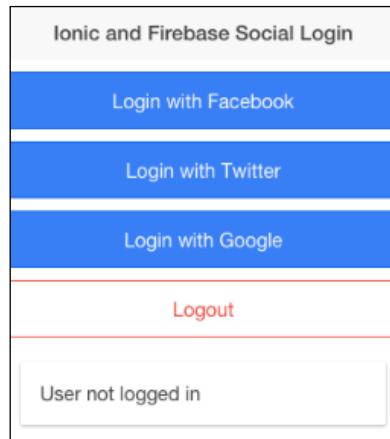
In this chapter, we will cover the following tasks related to user management:

- ▶ Configuring a Facebook app with the Firebase authentication
- ▶ Configuring a Twitter app with the Firebase authentication
- ▶ Configuring a Google+ project with the Firebase authentication
- ▶ Creating an Ionic social authentication project for Facebook using `$firebaseAuth`
- ▶ Creating a LinkedIn app and configuring the authentication in Auth0
- ▶ Integrating Auth0's LinkedIn authentication in an Ionic project

## Introduction

In this chapter, we will create three sample authentication apps:

- ▶ Social Authentication (Facebook, Twitter, and Google+) using Firebase



- ▶ Social Authentication (LinkedIn) using Auth0 and Firebase



Depending on the app, you may not need to use all of these authentication methods. For example, it would make more sense to use a LinkedIn authentication for an app focusing on a working professional to narrow down the audiences who fit the user profile of the app. Firebase supports many types of authentication with the exception of LinkedIn. Therefore, the LinkedIn authentication app will use a combination of Auth0 and Firebase.

The list of authentications supported in Firebase includes the following:

- ▶ Email and password
- ▶ Facebook
- ▶ Twitter
- ▶ GitHub
- ▶ Google
- ▶ Anonymous
- ▶ Custom

If you have your own authentication server where you maintain your own user database, you still can use the custom authentication of Firebase to create a custom token. You will see more details when you go through the LinkedIn authentication example, where Auth0 will ask Firebase for a custom token because Auth0 works with LinkedIn directly to authenticate users. You can use Auth0 for other authentication methods as well. It's really up to your app architecture. However, this chapter will try to simplify the authentication concept as much as possible. The sample apps will use Firebase as much as possible as a backend mechanism.

## Configuring a Facebook app with Firebase authentication

Most of the time, your users already have a social media account somewhere. It's more convenient for them to log into your app using the same social account instead of filling out the same information again. Firebase provides an authentication feature to reduce the need to build a Facebook authentication module in AngularJS from scratch. As an app developer, you receive several benefits. The following are some of the advantages:

- ▶ An easier login option for users
- ▶ An increased registration conversion rate
- ▶ More information about users with their Facebook accounts
- ▶ Faster time to market as you don't have to build your own authentication mechanism

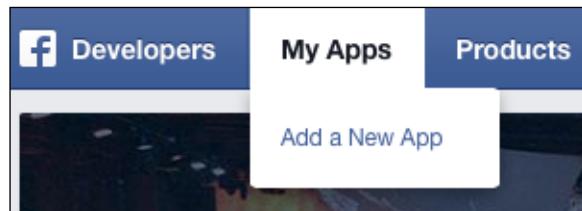
### Getting ready

You need to have a Facebook account to log in and create a developer app. The account must be verified using a phone number or credit card number. You must make sure that there is a way to verify your account. A Google Voice phone number will not be allowed. Facebook will ask for account verification when you create the app. Hence, you don't have to do this up front.

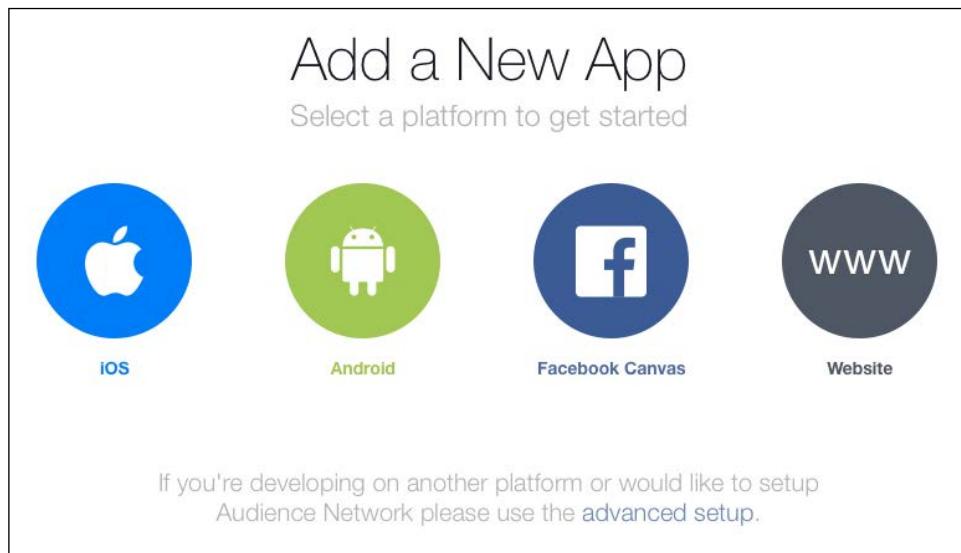
## How to do it...

In order for Firebase to authenticate users via the Facebook login, you have to create and configure the Facebook app to allow authentication and callback, as follows:

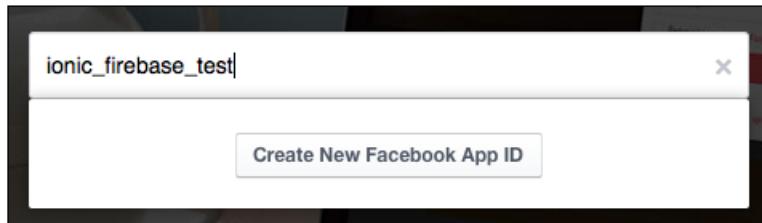
1. Log in to your Facebook account.
2. Navigate to <https://developers.facebook.com>.
3. Select **Add a New App** under the **My Apps** menu:



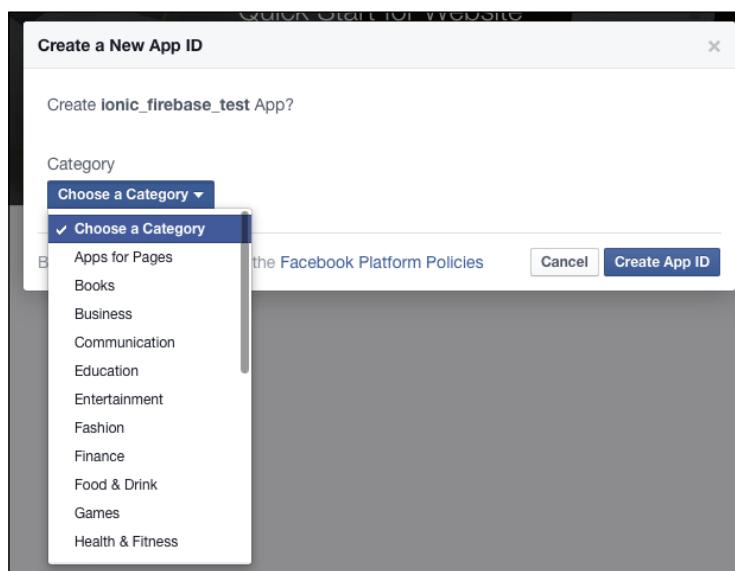
4. Select the **Website** button because we will be using JavaScript:



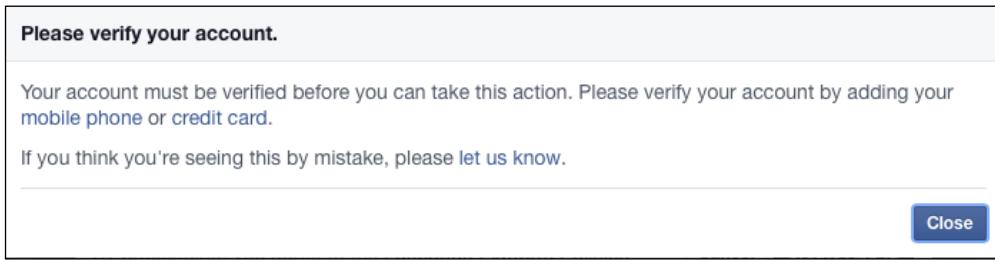
5. Provide the name of your app and click on the **Create New Facebook App ID** button.  
In this example, let's call it the `ionic_firebase_test` app:



6. Choose any category that you want from the drop-down menu and click on **Create App ID**:



7. If you have not verified your Facebook account, at this step, Facebook will ask for verification. You can either add a phone number or credit card number to pass this step:



8. After the verification, go to the **Settings** menu in the left panel.
9. Click on the **Advanced** tab and change your app restrictions or security as needed for your app:

Basic      Advanced      Migrations

Native or desktop app?  
Enable if your app is a native or desktop app

Deauthorize Callback URL  
What should we ping when a user deauthorizes your app?

10. Under **OAuth Settings**, fill in the **Valid OAuth redirect URIs** input box with your own URL using the `https://auth.firebaseio.com/v2/<YOUR-FIREBASE>/auth/facebook/callback` format. So, if your Firebase app's name is ionicebook, it will look like what is shown in the following screenshot:

OAuth Settings

Client OAuth Login  
Enables the OAuth client login flow

Embedded browser OAuth Login  
Browser control for OAuth client login

Valid OAuth redirect URIs  
https://auth.firebaseio.com/v2/ionicebook/auth/facebook/callback

11. Scroll down to the end of the page and click on the **Save Changes** button.

This is the basic app configuration that you need to do for Facebook. Now, you will have to modify Firebase to enable the Facebook authentication, as follows:

12. While still in the same Facebook app screen, click on the **Basic** tab and select the **Show** button to reveal the App Secret string:

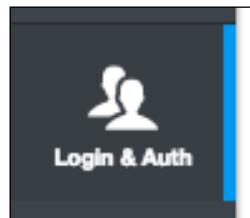
Basic      Advanced      Migrations

App ID  
1579401235659574

App Secret  
2adc2ec2aa7e1fd7c122bb5f216347aa

Reset

13. Copy both the App ID and the App Secret string because you need to import them to Firebase.
14. Log in to your Firebase account and select your app (for example, ionicebook).
15. Click on the **Login & Auth** menu, which looks like this:



16. Click on the **Facebook** tab.
17. Provide the same Facebook App ID and App Secret string that you copied earlier:

A screenshot of the Firebase Authentication settings page. At the top, there are tabs for 'Email & Password', 'Facebook' (which is selected and highlighted in blue), 'Twitter', 'GitHub', 'Google', 'Anonymous', and 'Custom'. Below the tabs, there is a section for 'Enable Facebook Authentication' with a checked checkbox. Underneath it, there are fields for 'Facebook App Id:' containing '1579401235659574' and 'Facebook App Secret:' containing '2adc2ec2aa7e1fd7c122bb5f216347aa'.

18. Check off the **Enable Facebook Authentication** checkbox, and everything will be saved automatically.

### How it works...

Firebase makes it very easy for you to work with the Facebook authentication because it takes care of server communication and builds in an API within AngularFire. You will explore the code in detail in the next section.

The authentication mechanism only needs the following three pieces of information:

- ▶ The Firebase Callback URL, so that Facebook knows how it should handle the redirect
- ▶ The Facebook App ID
- ▶ The Facebook Secret ID

You can trigger an authentication in the frontend simply by calling `$authWithOAuthPopup`.

### There's more...

Firebase has a very good documentation on authentication that uses either an email and a password, or social accounts. Just visit <https://www.firebaseio.com/docs/web/guide/user-auth.html> for more information.

If you want to go ahead and dive into the code, Firebase also provides an interactive demo, which is hosted on <http://jsfiddle.net/firebase/a221m6pb/>.

## Configuring a Twitter app with Firebase authentication

To authenticate using Twitter, Firebase also requires API Key and API Secret from your Twitter app. Firebase will take care of the communication and return the user object.

### Getting ready

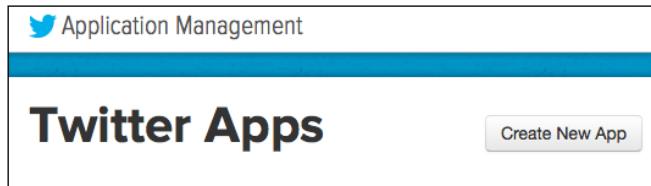
This is a straightforward process if you have already gone through the Facebook integration. It's similar to Twitter.

### How to do it...

Here are the instructions to set up your Twitter app:

1. Log in to your Twitter account.
2. Navigate to <https://apps.twitter.com>.

3. Click on the **Create New App** button:



Provide the application details. You can fill in anything for description and website as Twitter will not check for a test app. Let's name the app as `ionic_firebase_test`. The most important field here is the **Callback URL**, as it must be in the `https://auth.firebaseio.com/v2/<YOUR-FIREBASE>/auth/twitter/callback` format. Since the Firebase app name is `ionicebook`, the callback URL will be `https://auth.firebaseio.com/v2/ionicebook/auth/twitter/callback`, as shown in the following screenshot:

The screenshot shows a form titled "Application Details". It contains four main sections: "Name", "Description", "Website", and "Callback URL".

- Name \***: The input field contains "ionic\_firebase\_test".  
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.
- Description \***: The input field contains "Test Twitter Authentication for Ionic and Firebase".  
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.
- Website \***: The input field contains "http://www.PlaceholderIsOK.com".  
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.  
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)
- Callback URL**: The input field contains "https://auth.firebaseio.com/v2/ionicebook/auth/twitter/callback".  
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Allow this application to be used to [Sign in with Twitter](#)

## User Registration and Authentication

---

4. Check off the **Allow this application to be used to Sign in with Twitter** checkbox.
5. Check off the **Yes, I agree** checkbox for **Developer Agreement**:

**Developer Agreement**

New! We revised our Developer Agreement, effective as of May 18, 2015. Please read the [updated Developer Agreement](#). By continuing to access or use our content or services after May 18, 2015, you agree to the revisions.

Last Update: October 22, 2014.

This Twitter Developer Agreement ("Agreement") is made between you (either an individual or an entity, referred to herein as "you") and Twitter, Inc., on behalf of itself and its worldwide affiliates (collectively, "Twitter") and governs your access to and use of the Licensed Material (as defined below).

PLEASE READ THE TERMS AND CONDITIONS OF THIS AGREEMENT CAREFULLY, INCLUDING WITHOUT LIMITATION ANY LINKED TERMS AND CONDITIONS APPEARING OR REFERENCED BELOW, WHICH ARE HEREBY MADE PART OF THIS LICENSE AGREEMENT. BY USING THE LICENSED MATERIAL, YOU ARE AGREEING THAT YOU HAVE READ, AND THAT YOU AGREE TO COMPLY WITH AND TO BE BOUND BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND ALL APPLICABLE LAWS AND REGULATIONS IN THEIR ENTIRETY WITHOUT LIMITATION OR QUALIFICATION. IF YOU DO NOT AGREE TO BE BOUND BY THIS AGREEMENT, THEN YOU MAY NOT ACCESS OR OTHERWISE USE THE LICENSED MATERIAL. THIS AGREEMENT IS EFFECTIVE AS OF THE FIRST DATE THAT YOU USE THE LICENSED MATERIAL ("EFFECTIVE DATE").

Yes, I agree

[Create your Twitter application](#)

6. Click on the **Create your Twitter application** button.
7. Navigate to the **Keys and Access Tokens** tab:



8. Under **Application Settings**, copy the API Key and API Secret:

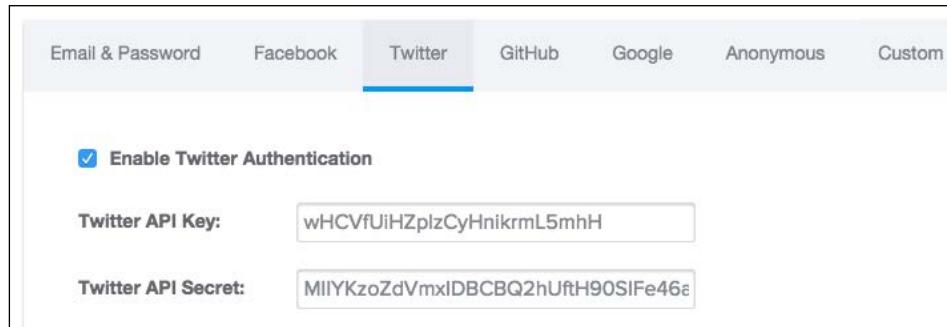
## Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	wHCVfUiHZplzCyHnikrmL5mhH
Consumer Secret (API Secret)	MIIYKzoZdVmxDDBCQ2hUftH90SIFe46ap73cfJxJg0U9bE6n7

You will need to provide Firebase the Twitter API Key and API Secret. Here are the steps:

9. Log in to your Firebase account and select your app (for example, ionicebook).
10. Click on the **Login & Auth** menu.
11. Click on the **Twitter** tab.
12. Provide the same Twitter API Key and API Secret that you copied earlier:



13. Check off the **Enable Twitter Authentication** checkbox, and everything will be saved automatically.
14. Don't close the **Firebase** tab or browser window, as you will be required to come back to provide information for the Google authentication later.

## How it works...

The steps that we went through are used to set up the Twitter app correctly. If you already have an existing Twitter app, you can just use the API Key and API Secret from this app instead. As you have already seen, configuring on the side of Firebase to enable the Twitter authentication only takes two parameters. That's it.

## Configuring a Google+ project with Firebase authentication

Google has a very comprehensive list of cloud services. Also, it's a good idea to include Google as your authentication mechanism due to a large number of users. In addition, Firebase is a Google company. Therefore, in the future, this process can be a lot simpler and you may have access to additional features.

## Getting ready

If you don't already have a Google test account, you should create one.

## How to do it...

Here are the instructions that are required to set up your Google+:

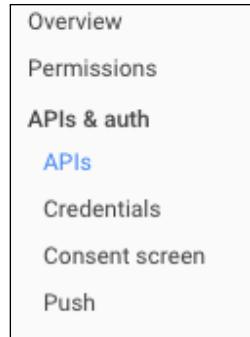
1. Navigate to <https://console.developers.google.com>.
2. Log in to your Google account.
3. Click on the **Create Project** button, as Google calls it a *project* rather than an app:



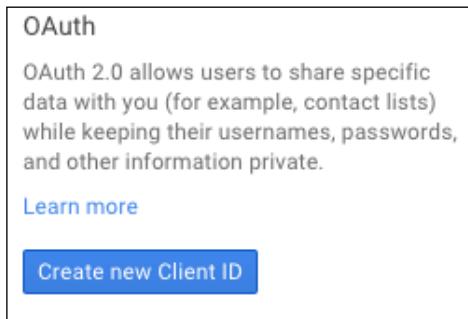
4. Fill in the name as `IonicFirebaseTest`, since Google does not allow the usage of a hyphen or an underscore:

A screenshot of the "New Project" dialog box. It has a title "New Project". Inside, there are two input fields: "Project name" containing "IonicFirebaseTest" and "Project ID" containing "steam-state-93106". Below these fields is a link "Show advanced options...". At the bottom are two buttons: a blue "Create" button and a white "Cancel" button.

5. Navigate to **APIs & auth | APIs** from the menu to the left:



6. Under **OAuth**, click on the **Create new Client ID** button:



7. Provide the client ID details. Make sure that web application is checked, as we use JavaScript for the app. The **Authorized JavaScript origins** text box should have `https://auth.firebaseio.com`. The **Authorized redirect URIs** text box will have `https://auth.firebaseio.com/v2/ionicebook/auth/google/callback`, as the Firebase app name is ionicebook:

Create Client ID

**Application type**

Web application  
Accessed by web browsers over a network.

Service account  
Calls Google APIs on behalf of your application instead of an end-user. [Learn more](#)

Installed application  
Runs on a desktop computer or handheld device (like Android or iPhone).

**Authorized JavaScript origins**  
Cannot contain a wildcard (`http://*.example.com`) or a path (`http://example.com/subdir`).  
`https://auth.firebaseio.com`

**Authorized redirect URIs**  
One URI per line. Needs to have a protocol, no URL fragments, and no relative paths. Can't be a non-private IP Address.  
`https://auth.firebaseio.com/v2/ionicebook/auth/google/callback`

**Create Client ID** **Cancel**

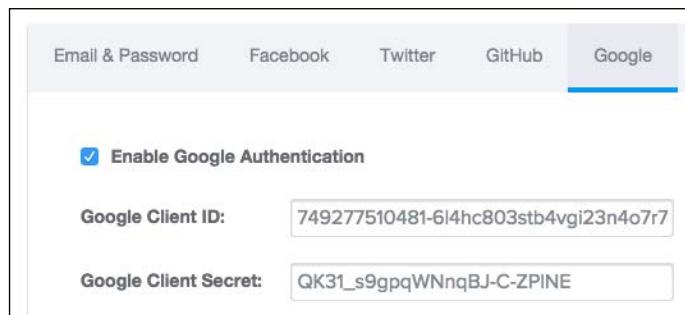
8. Click on the **Create Client ID** button to save the settings.
9. Under **Client ID for web application**, copy the client ID and client secret, which is used for Firebase:

Client ID for web application	
Client ID	749277510481-6l4hc803stb4vgj23n4o7r7lttrg1gmt.apps.googleusercontent.com
Email address	749277510481-6l4hc803stb4vgj23n4o7r7lttrg1gmt@developer.gserviceaccount.com
Client secret	QK31_s9gpqWNnqBJ-C-ZPINE
Redirect URIs	<code>https://auth.firebaseio.com/v2/ionicebook/auth/google/callback</code>
JavaScript origins	<code>https://auth.firebaseio.com</code>

**Edit settings** **Reset secret** **Download JSON** **Delete**

Similarly, you will need to provide Firebase the Google client ID and client secret. The following are the steps that are required to accomplish this:

1. Log in to your Firebase account and select your app (for example, ionicebook).
2. Click on the **Login & Auth** menu.
3. Select the **Google** tab.
4. Provide the same Google client ID and client secret that you copied earlier:



5. Check off the **Enable Google Authentication** checkbox, and everything will be saved automatically.

### How it works...

There is no difference in terms of how Google, Facebook, and Twitter work for the OAuth authentication. However, note that the redirect or callback URLs for each method are a bit different. This is because Firebase has a different backend mechanism to handle each authentication endpoints.

## Creating an Ionic social authentication project for Facebook using \$firebaseAuth

So far, you have been just configuring the apps. This recipe will explain how easy it is to write the code to authenticate a user via Facebook, Twitter, or Google. You will go through an example app to set up a **login** button. The user will click on **login** to authenticate via his or her social network. Then, the app will display the basic profile information, including the user's picture. Finally, the user can click on **logout** to end the session.

## Getting ready

The only thing that you need here is the Ionic CLI to perform the steps.

## How to do it...

The following instructions are used to code for the Facebook authentication:

1. Create a blank Ionic app (for example, SocialAuth), as follows:

```
$ ionic start SocialAuth blank
```

2. Go to the project folder:

```
$ cd SocialAuth
```

3. For authentication to work on a physical device, it will need two Cordova plugins, InAppBrowser and Cookies. The reason is that the authentication popup window will show in another browser window. Also, for Firebase to work, it needs native cookie support in the app. Type the following two command lines in your app folder:

```
$ cordova plugin add org.apache.cordova.inappbrowser
```

```
$ cordova plugin add https://github.com/bez4pieci/Phonegap-Cookies-Plugin.git
```

4. Open the index.html file under /www to edit the main template.

5. Install the Firebase and AngularFire modules:

```
$ sudo bower install angularfire -save --allow-root
```

6. Add the Firebase and AngularFire modules:

```
<!-- Firebase -->
<script src="lib.firebaseio.firebaseio.js"></script>
```

```
<!-- AngularFire -->
<script src="lib/angularfire/dist/angularfire.js"></script>
```

7. Under <body ng-app="starter">, replace with the following code because you will show four buttons—**Facebook**, **Twitter**, **Google**, and **Logout**. Then, after logging in, the UI will display some basic user profile in the user object:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
```

```
<ion-header-bar>

<ion-content ng-controller="LoginCtrl">
  <!-- Buttons -->
  <button class="button button-full button-positive"
    ng-click="login(providerConf[0])">Login with
    Facebook</button>
  <button class="button button-full button-positive"
    ng-click="login(providerConf[1])">Login with
    Twitter</button>
  <button class="button button-full button-positive"
    ng-click="login(providerConf[2])">Login with
    Google</button>
  <button class="button button-full button-outline
    button-assertive" ng-click="logout()">Logout</button>

  <!-- Logged-in user text -->
  <div class="card" ng-hide="user">
    <div class="item item-text-wrap">User not
      logged in</div>
    <!-- <div class="item item-text-wrap"
      ng-bind="user.uid"></div> -->
  </div>
  <div class="list" ng-show="user">
    <a class="item item-thumbnail-left" href="#">
      
      <h2>{{ user.name }}</h2>
      <p>{{ user.uid }}</p>
      <p>{{ user.email }}</p>
    </a>
  </div>
</ion-content>
</ion-pane>
```

8. Open app.js under /www/js to edit it.
9. Make sure that you include firebase in the module:

```
var app = angular.module("starter", ["ionic", "firebase"]);
```
10. Keep app.run() the same, as that will not change for this project. However, delete all the other boilerplate if it exists. You will create the Auth factory and the LoginCtrl controller.

11. Add the Auth factory as follows because you will need to call `login()` and `logout()` and get the user profile data via `onAuth()`. Calling `Firebase()` with your Firebase URL will create a reference object for you to work with:

```
// create a custom Auth factory to handle $firebaseAuth
app.factory('Auth', function($firebaseAuth, $timeout) {
  var ref = new
  Firebase('https://ionicebook.firebaseio.com');
  var auth = $firebaseAuth(ref);

  return {
    // helper method to login with multiple providers
    login: function (provider, options) {
      var result;
      if (options)
        result = auth.$authWithOAuthPopup(
          provider, {scope: options});
      else
        result = auth.$authWithOAuthPopup(provider);
      return result;
    },
    // wrapping the unauth function
    logout: function () {
      auth.$unauth();
    },
    // wrap the $onAuth function with $timeout so it
    processes
    // in the digest loop.
    onAuth: function (callback) {
      auth.$onAuth(function(authData) {
        $timeout(function() {
          callback(authData);
        });
      });
    }
  };
});
```

12. Create the `LoginCtrl` controller with some initial values in `$scope`, as follows:

```
app.controller("LoginCtrl", function($scope, Auth) {
  // Initially set no user to be logged in
  $scope.user = null;
```

```
// Assign permission request per provider
$scope.providerConf = [
  {
    name: "facebook",
    options: "email,user_likes,
    publish_actions,user_about_me,read_stream"
  },
  {
    name: "twitter"
    // Twitter has no permission object so
    we don't have to pass the options string
  },
  {
    name: "google",
    options: "profile,email,openid"
  },
];
});
```

13. Within the `LoginCtrl` controller, add the `login()` function to `$scope` so that the frontend can access it. This is done so that the three login buttons just pass the proper provider string to trigger the login via Firebase:

```
// Calls $authWithOAuthPopup on $firebaseAuth
// This will be processed by the InAppBrowser
// plugin on mobile
// We can add the user to $scope here or in the $onAuth fn
$scope.login = function scopeLogin(providerObj) {

  Auth.login(providerObj.name, providerObj.options)
  .then(function(authData) {
    console.log('We are logged in ' +
    providerObj.name + ' !', authData);
  })
  .catch(function(error) {
    console.error(error);
  });
};
```

14. Assign the `logout()` function straight from the factory. This is also for the `logout` button to call:

```
// Logs a user out
$scope.logout = Auth.logout;
```

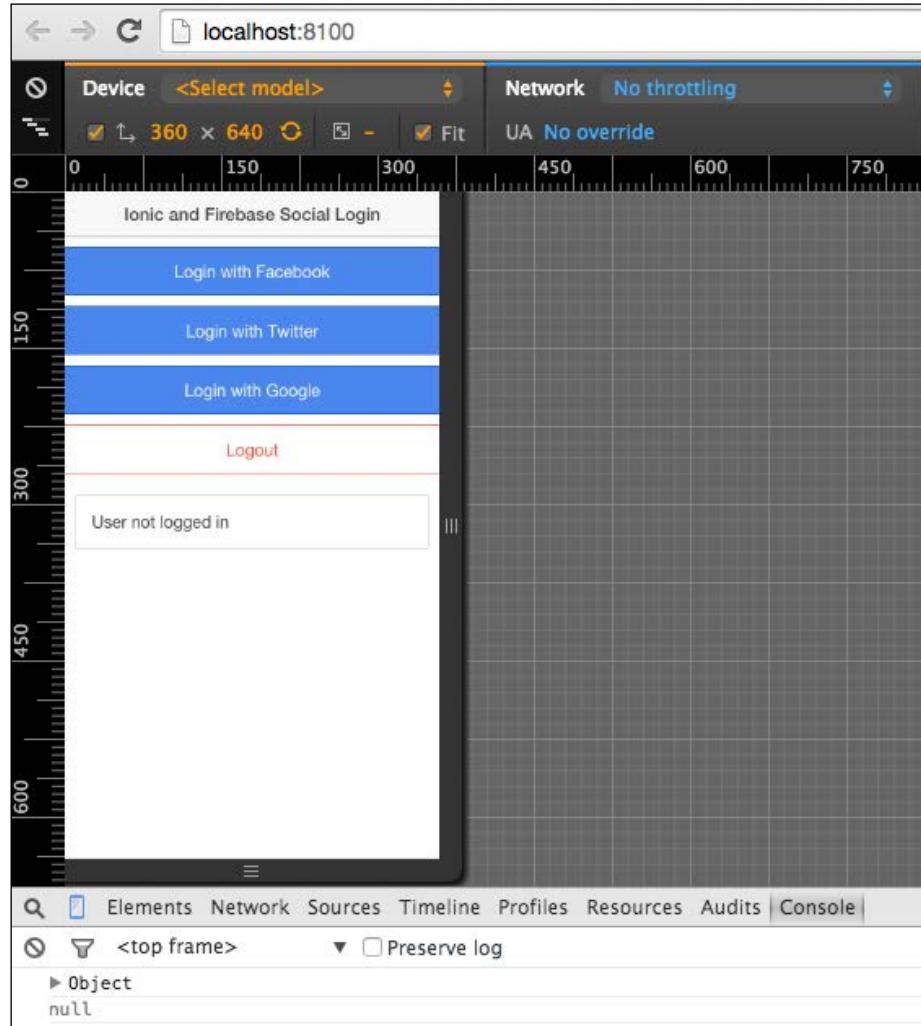
15. Finally, you need to customize the user object based on whether it's a Facebook, Twitter, or Google login. The main reason behind this is that each method returns its own social network profile object. You want a uniform way in order to render the user profile correctly in the template:

```
// detect changes in authentication state
// when a user logs in, set them to $scope
Auth.onAuth(function(authData) {
  $scope.user = authData;
  console.log(authData);
  if ($scope.user) {
    switch ($scope.user.provider) {
      case "facebook":
        $scope.user.avatar = authData.facebook
          .cachedUserProfile.picture.data.url;
        $scope.user.name = authData.facebook.displayName;
        $scope.user.email = authData.facebook.email;
        break;
      case "twitter":
        $scope.user.avatar = authData.twitter
          .cachedUserProfile.profile_image_url;
        $scope.user.name = authData.twitter.displayName;
        $scope.user.email = authData.twitter
          .cachedUserProfile.description;
        break;
      case "google":
        $scope.user.avatar = authData
          .google.cachedUserProfile.picture;
        $scope.user.name = authData.google.displayName;
        $scope.user.email = authData.google.email;
        break;
    }
  }
});
```

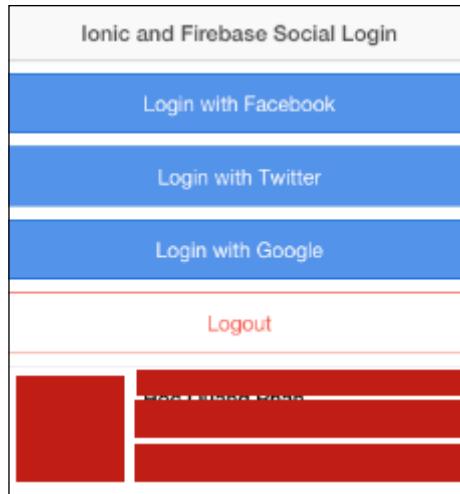
16. Go back to the console and test in your browser. Make sure that you run the following command line in your project folder and not the /www folder:

```
$ ionic serve
```

17. You should be able to see the login screen, as shown in the following screenshot:



18. When you click on the **Facebook** button and log in, you should be able to see your avatar, name, and email:



19. Take a look at the console and see more details of the user object. You can expand on the app and use any data that you like.

Note that for production deployment, you may want to remove all the `console.log()` outputs because it's not a good idea to reveal information to the end users (if they decided to turn on the debug functionality in a browser):



## How it works...

The key for the social login to work is calling `$authWithOAuthPopup`. You can pass a provider string (such as "facebook", "twitter", and "google") for this and indicate which social network you want to use. Firebase will take care of the entire login process and return the user's profile data in an `authData` object. You can do anything you want with this object.

In the preceding example, the code will take some important fields (`email`, `avatar`, `ID`, and `name`) and assign them to the `$scope.user` object. This is how Angular detects and updates the template on the frontend. Each social network has its own object structure. That's why you have to detect it via `switch ($scope.user.provider) {}` (or if-else) and properly get the value from the `authData` object.

### There's more...

For testing and development, you can request for any permission. However, if you want to publish the Facebook app in production, you need to go through a review process. This may take a few days. For more information on this, visit <https://developers.facebook.com/docs/facebook-login/permissions/v2.5>, the Facebook developer website.

## Creating a LinkedIn app and configuring authentication in Auth0

Firebase does not support LinkedIn natively, but it does allow the creation of a custom token. This method is used mainly when the authentication mechanism is actually performed somewhere else, either on your own server or another SaaS provider. In this case, you will be using Auth0 to provide the LinkedIn authentication. You could have performed the Facebook, Twitter, or Google authentication via Auth0 as well. However, it's recommended to use Firebase as much as possible as it's simpler to use. This is not only due to cost-related reasons, but also for the reduction of the number of integration points and code complexity.

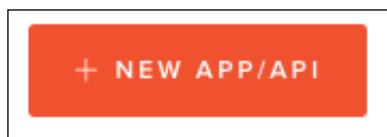
### Getting ready

Account registration is required for Auth0, Firebase, and LinkedIn in order to perform the steps in this recipe.

### How to do it...

You need to configure your app in Auth0 first by performing the following steps:

1. Sign up for an Auth0 account and navigate to <https://manage.auth0.com/>.
2. You will see the dashboard first, but there is nothing there yet because there is no app defined. To create an app, click on the **New App/API** button:



3. Give the app a name, as follows:

The screenshot shows a 'Create Application' dialog box. At the top, it says 'Create Application'. Below that, there is a 'Name' field containing the text 'Linkedin Login'. Underneath the name field is a placeholder text: 'This is a friendly name for your application.' At the bottom right of the dialog box is a blue 'SAVE' button.

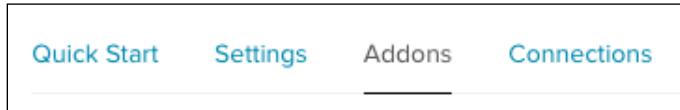
4. You can ignore the first **Quick Start** tab because it's better to learn through specific details inside the app. Select the **Setting** tab and fill out the form. Let's assume that we used ionicebook as the domain name when creating the account. The callback URL will be `https://ionicebook.auth0.com/mobile` and **Allowed Origins (CORS)** will have the value `file:///*, http://localhost:8100/`. The reason behind this is that you have to tell Auth0 that it is OK to get the REST calls from these URLs. If you run the app in your physical device, it will call from a file system (`file://`). If you run the URL via a browser from a local server (`ionic serve`), it will be `http://localhost:8100/`:

The screenshot shows the 'Settings' tab configuration for an application. The fields filled in are:

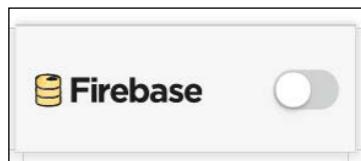
- Domain: ionicebook.auth0.com
- Client ID: eVuc9hYlogywIBFmSA89QuO2bif6oC3J
- Client Secret: ePmXQnjZm1li0BxQ3iHZJAC1JfewaTMiqzj3GYi
- Allowed Callback URLs: https://ionicebook.auth0.com/mobile
- Allowed Origins (CORS): file:///\*, http://localhost:8100/
- JWT Expiration (seconds): 36000
- Use Auth0 instead of the IdP to do Single Sign On:

At the bottom left is a 'Show Advanced Settings' link, and at the bottom right is a blue 'SAVE CHANGES' button.

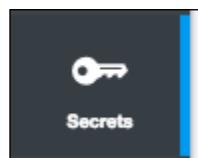
5. Navigate to the **Addons** tab:



6. Select the Firebase icon:



7. A dialog will appear for you to provide the Firebase Secret. If you don't have that ready, go to the Firebase app and navigate to the **Secrets** tab:



8. Click on the **Show** button below the input box and then copy the whole secret string, as shown in the following screenshot:



## User Registration and Authentication

---

9. Go back to the Auth0 app and paste to the LinkedIn add-on:

4. Copy the **Secret** and paste it here:

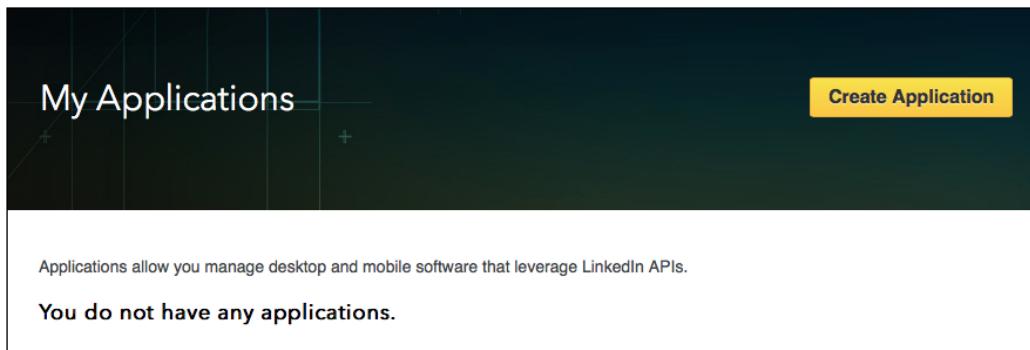
**Firebase Secret**      o4LrKyVDGf8STb8lI7CJV9J9CbTcliS3T2IS6!      **SAVE**

Get the secret from your Firebase dashboard.

10. Click on **Save**.

That's it as regards to configuring Auth0 and Firebase. However, you still need to connect to LinkedIn. Before changing anything in Auth0, you must get the LinkedIn API and Secret Key to provide to Auth0. The following are the steps that are required to set up a LinkedIn app:

1. Create a LinkedIn account if you don't already have one and log in.
2. Navigate to <https://www.linkedin.com/developer/apps>.
3. Click on the **Create Application** button:



4. Fill out the form as shown in the following screenshot. The most important part is that you need to have Website URL as either `http://localhost:8100`, or whatever URL you will be using to test the app. You can change this later once it's in production:

**Create a New Application**

**Company Name:**\*

**Name:**\*

**Description:**\*

**Application Logo URL:**\*

**Application Use:**\*

**Website URL:**\*

**Business Email:**\*

**Business Phone:**\*

I have read and agree to the [LinkedIn API Terms of Use](#).

**Submit** **Cancel**

5. Make sure that you check off **Terms of Use** and click on the **Submit** button.
6. Navigate to the **Authentication** menu to the left. It should be available on the first page.

7. Copy the client ID and client Secret. You will use those for the Auth0 inputs:

The screenshot shows a box titled "Authentication Keys". Inside, there are two fields: "Client ID: 75stubnn1n6tyg" and "Client Secret: c2KohxC8xIOJfoEc".

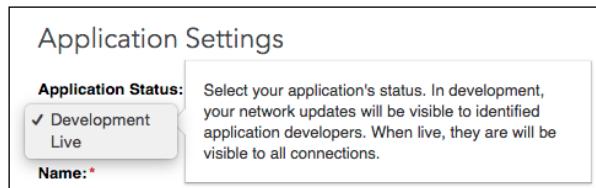
8. Make sure that you check off the permission that you want to acquire from the users. In this example, let's check off all the checkboxes so that you can see what LinkedIn returns in the authentication:

The screenshot shows a box titled "Default Application Permissions" containing a grid of checkboxes. All checkboxes are checked:  
r\_basicprofile, r\_contactinfo, r\_emailaddress  
r\_fullprofile, r\_network, rw\_company\_admin  
rw\_groups, rw\_nus, w\_messages  
w\_share

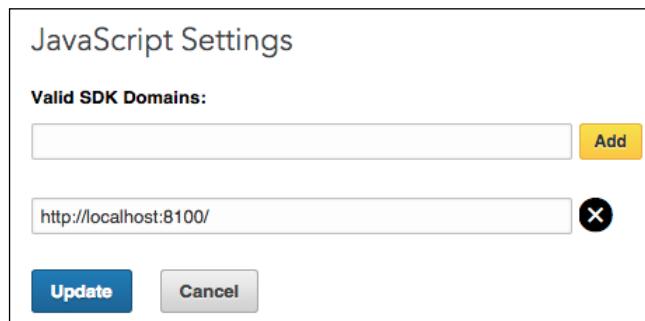
9. In order for the authentication process to know the callback, you need to provide that information in the OAuth inputs. It will be your Auth0 URL, with the trailing /login/callback instance in the URL:

The screenshot shows a form for OAuth configuration. The top part is for OAuth 2.0, showing the "Authorized Redirect URLs" field with "https://ionicebook.auth0.com/login/callback" and an "Add" button. Below it is the "X" button to remove the URL. The bottom part is for OAuth 1.0a, showing the "Default 'Accept' Redirect URL" field with "https://ionicebook.auth0.com/login/callback" and the "Default 'Cancel' Redirect URL" field below it. At the bottom are "Update" and "Cancel" buttons.

10. Navigate to the **Settings** tab from the menu to the left and ensure that **Application Status** is **Development**. You will need to switch this to **Live** when you go into production:



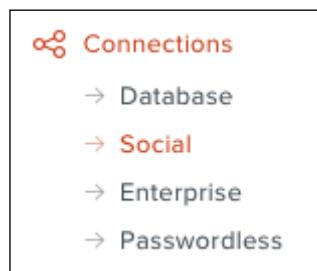
11. Navigate to the **JavaScript** tab and make sure that **Valid SDK Domains** is `http://localhost:8100/` or whatever local URL for development:



12. Click on the **Update** button to finish configuring your LinkedIn app.

At this point, you have completed setting up Auth0, connecting it to Firebase and creating your own LinkedIn app. However, there is one more thing that is required to complete the process, and that thing is the connection of the Auth0 and LinkedIn app. Otherwise, Auth0 will not know where to send the data to. This can be done in a few simple steps:

1. Navigate to the Auth0 app and browse to **Connections | Social** from the menu to the left:



2. Click on the **Linkedin** icon:



3. Fill in the API Key and Secret Key that you copied earlier from the LinkedIn app. Make sure that you check off the attributes and permissions that you want. In this example, you can check off all of them to review the kind of data that will be returned from LinkedIn:

The form is titled "LinkedIn". It has two tabs: "Settings" (selected) and "Apps".

**Name:** linkedin

**API Key:** 75stubnn1n6tyg  
[How to obtain an API Key?](#)

**Secret Key:** c2KohxC8xIOJfoEc

**Attributes:**

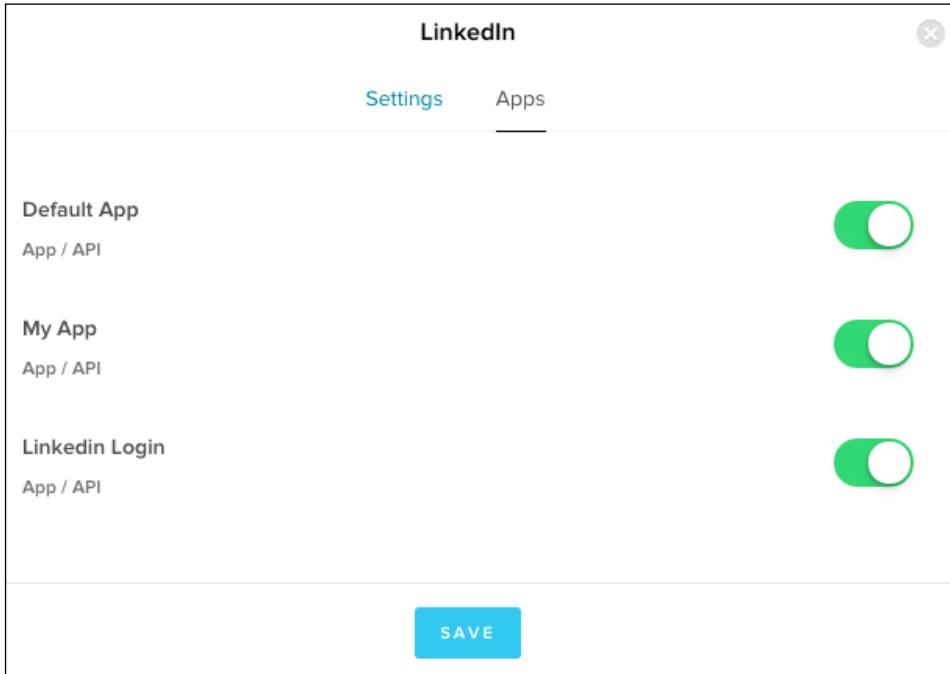
- Basic Profile required
- Full Profile
- Email address

**Permissions:**

- Network

**Buttons:** SAVE (blue button), TRY (orange button)

4. Navigate to the **Apps** tab in the menu at the top:



5. Make sure that your Auth0 app (such as **LinkedIn Login**) is checked off with the green toggle button.
6. Click on **Save**.

### How it works...

So far, you have not written a single line of code. This is because Auth0 and Firebase will take care of the backend data exchange so that the user will be authenticated via LinkedIn. In the next section, you will take a look at how simple it is to actually authenticate a user and get the returned user object.

The way it works in this setup is as follows:

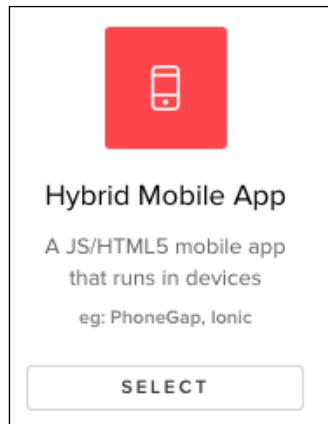
1. Auth0 acts as a trigger to call the LinkedIn API to exchange user data.
2. Once LinkedIn says, "**Okay, this user is authenticated**", Firebase will get the object and store it in the memory. At the same time, Auth0 will ask Firebase to create a custom token to pass data to Firebase.
3. Firebase will then agree to authenticate on its own backend and allow the user to get data from the Firebase database.
4. Ionic will act as the frontend entirely and only render the data, as returned from the APIs. You will explore more possibilities in this area later on.

### There's more...

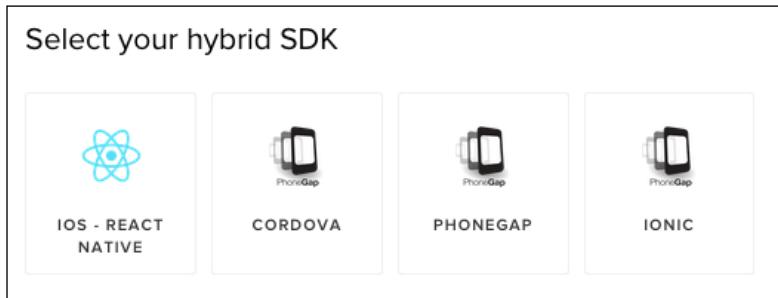
Auth0 made it easy by providing you with a wizard depending on how you build the app (for instance, by using an add-on or a third-party plugin). You can download the *seed project* from Auth0. However, you need to be aware that the app tends to have more boilerplate than the most basic code, which will be discussed in the next recipe. The seed project has a full-blown navigation with routes and templates. You will explore more about those topics in other chapters.

In case you want to use the seed project, the following are the steps:

1. Go to your app page and under the **Quick Start** tab, let's select **Hybrid Mobile App** because Auth0 has specific steps defined for Ionic and Firebase already:



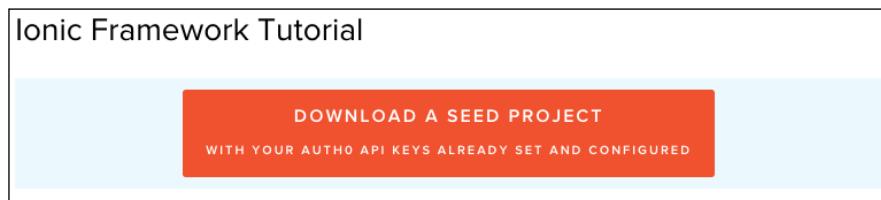
2. Next, select **Ionic** in the SDK list:



3. Select **Firebase** under **3rd Party API**:



4. Click on the **Download A Seed Project** button:



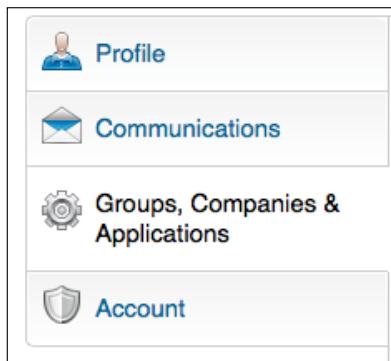
It will be useful to know where the Auth0 app is being authorized within the test user LinkedIn account. During testing and development, you may have to manually remove this authorization once in awhile. The reason behind this is that there may be some cache or bugs. Removing the authentication in the test user will let you start from scratch.

The following is the process that provides the details of how to do this:

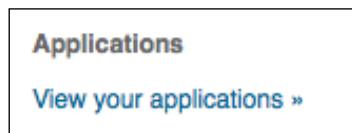
1. Log in as a LinkedIn test user (not the LinkedIn account that has the app, even though it can be the same).
2. In the top-right menu, navigate to **Privacy & Settings**:



3. Navigate to **Groups, Companies & Applications** from the menu to the left:



4. Click on the **View your applications** link:



5. If you have been testing the app, you will see the app name in the list. All you need to do is to check off the app name and click on **Remove**:

**Authorized External Applications**

Listed here are external partner applications to which you have granted access to your LinkedIn profile and network data. If you remove that access here, they will no longer be able to access your LinkedIn data. To re-enable them in the future, go to the application and grant access again.

Partner Name
<input type="checkbox"/> Buffer
<input type="checkbox"/> OS X Mavericks
<input type="checkbox"/> LinkedIn Help Center - Customer Portal
<input checked="" type="checkbox"/> IonicEbook

**Remove or Cancel**

6. After removing the app, the clean list should not have your Auth0 app name any more. Your list could look different from the one shown in the following screenshot, depending on the other apps that you have authorized in the past:

**Authorized External Applications**

Listed here are external partner applications to which you have granted access to your LinkedIn profile and network data. If you remove that access here, they will no longer be able to access your LinkedIn data. To re-enable them in the future, go to the application and grant access again.

Partner Name
<input type="checkbox"/> Buffer
<input type="checkbox"/> OS X Mavericks
<input type="checkbox"/> LinkedIn Help Center - Customer Portal

**Remove or Cancel**

## Integrating Auth0's LinkedIn authentication in an Ionic project

Once you have set up Auth0, Firebase, and LinkedIn, the boilerplate code that is needed to authenticate a user is very simple. This is the beauty of using Auth0 for this process. In this recipe, you will learn how to go through an app development that is similar to that of the app that you created earlier for Twitter, Facebook, and Google. Basically, a user will click on **Login** to make an authentication dialog pop up. After being authenticated, the app will show the profile photo, name, unique ID, and email address.

### Getting ready

You can start from scratch by creating a blank Ionic app. It's faster to make a copy of the previous Firebase app and change a few lines of code.

### How to do it...

You only need to write code in two files, `index.html` and `app.js`:

1. You need to download the required modules from Auth0. All of them are located in Auth0's `auth0-ionic` example. Navigate to <https://github.com/auth0/auth0-ionic/tree/master/examples/firebase-sample/www/lib> and download `auth0-lock`, `auth0-angular`, `angular-jwt`, `firebase`, and `angularfire`.
2. Include these modules in `index.html`. It will look like this:

```
<script src="lib/auth0-lock/build/  
auth0-lock.js"></script>  
<!-- ionic/angularjs js -->  
<script src="lib/ionic/js/ionic.bundle.js"></script>  
  
<script src="lib/auth0-angular/build/  
auth0-angular.js"></script>  
<script src="lib/angular-jwt/dist/  
angular-jwt.js"></script>  
<script src="lib.firebaseio.firebaseio.js"></script>  
<script src="lib/angularfire/dist/  
angularfire.js"></script>
```

3. Just like in the social authentication app using Firebase, you will need a **Login** and **Logout** button and a place to display some basic user profile. Just change the HTML under the `<body>` tag to the following:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
    <h1 class="title">Linkedin Authentication</h1>
  </ion-header-bar>

  <ion-content ng-controller="LoginCtrl">
    <!-- Buttons -->
    <button class="button button-full button-positive"
      ng-click="login()">Login with Linkedin</button>
    <button class="button button-full button-outline
      button-assertive" ng-click="logout()">Logout</button>

    <!-- Logged-in profile text -->
    <div class="card" ng-hide="profile">
      <div class="item item-text-wrap">User not
        logged in</div>
    </div>
    <div class="list" ng-show="profile">
      <a class="item item-thumbnail-left" href="#">
        
        <h2>{{ profile.name }}</h2>
        <p>{{ profile.user_id }}</p>
        <p>{{ profile.email }}</p>
      </a>
    </div>
  </ion-content>
</ion-pane>
```

4. Open and edit `/www/js/app.js`.
5. You need a global variable to keep track of the session data. This is just for simplicity, as the best practice is to keep this in local storage or a cookie within `$scope`. More information on this will be discussed later.

```
var LINKEDIN = {};
```

6. Your app must include the modules that we downloaded earlier:

```
var app = angular.module('starter', [
  'ionic',
  'auth0',
  'angular-jwt',
  'firebase'
]);
```

7. There is no change within `app.run()`. So, let's leave it as is.
8. Unlike Firebase login, Auth0 requires some configuration in your AngularJS app. First, it initializes `authProvider` with your Auth0 information. Note that the `clientID` variable is the same client ID string from your Auth0 app. Then, you need to tell Auth0 to send the authentication token string each time an API is called to Firebase. That's why you set up the function in `jwtInterceptorProvider` and push it to `$httpProvider`:

```
app.config(function(authProvider, jwtInterceptorProvider,
$httpProvider) {
  // Configure Auth0
  authProvider.init({
    domain: 'ionicebook.auth0.com',
    clientID: 'eVuc9hYlogywlBFmSA89QuO2bif6oC3J',
    loginState: 'login'
  });

  jwtInterceptorProvider.tokenGetter = function(store,
  jwtHelper, auth) {
    var idToken = LINKEDIN.token;
    var refreshToken = LINKEDIN.refreshToken;
    if (!idToken || !refreshToken) {
      return null;
    }
    if (jwtHelper.isTokenExpired(idToken)) {
      return auth.refreshIdToken(refreshToken) .
      then(function(idToken) {
        LINKEDIN.token = idToken;
        return idToken;
      });
    } else {
      return idToken;
    }
  }
})
```

```
$httpProvider.interceptors.push('jwtInterceptor');  
});
```

9. You need to write the code for the `LoginCtrl` controller. This will have two functions: `login()` and `logout()`, as follows:

```
app.controller('LoginCtrl', function($scope, $rootScope,  
auth) {  
    $scope.login = function() {  
        auth.signin({  
            closable: false,  
            // This asks for the refresh token  
            // So that the user never has to log in again  
            authParams: {  
                scope: 'openid offline_access'  
            }  
        }, function(profile, idToken, accessToken, state,  
refreshToken) {  
            LINKEDIN.profile = profile;  
            $scope.profile = profile;  
            console.log('profile:');  
            console.log(profile);  
            LINKEDIN.token = idToken;  
            LINKEDIN.refreshToken = refreshToken;  
            auth.getToken({  
                api: 'firebase'  
            }).then(function(delegation) {  
                console.log('delegation:');  
                console.log(delegation);  
            }, function(error) {  
                console.log("There was an error logging in",  
error);  
            })  
        }, function(error) {  
            console.log("There was an error logging in", error);  
        });  
    }  
  
    $scope.logout = function() {  
        auth.signout();  
        LINKEDIN = {};  
        $scope.profile = undefined;  
    }  
});
```

10. There are many lines with `console.log()`. The goal is to output the entire object so that you can inspect its data. After saving both the files, you can test these on the browser, as follows:

```
$ ionic serve
```

11. Authenticate via the LinkedIn test user's account and open the browser console. You should be able to inspect the profile object and the delegation object that is passed to Firebase:

## How it works...

Note that the login modal is not from the current code. It was a ready-to-use template from Auth0 when you included `auth0-lock.js`. You don't have to use this template, but it's convenient for the purpose of this recipe. The `signin()` function does the majority of the work to authenticate the user and return the profile object. Then, you will need to use the `getToken()` function to encapsulate the data in a JSON object so that it can be given to Firebase. Later, we will also define how Firebase uses this information to create a custom token using `authWithCustomToken()` as well. The basic mechanism is that Firebase will receive all the profile data and use the information to properly decide upon the user permission using the security rule of Firebase. For example, your app only allows authenticated users to have access to their own data. Therefore, you need to map the email address to `userId` and ensure that `userId` is the same as `ownerId`.

### **See also**

This recipe does not cover local storage, because when the app is closed and the user comes back, they can still be authenticated automatically without having to log in to LinkedIn again. You will learn more about local storage using JavaScript and Cordova plugin in the later chapters of this book.

# 9

## Saving and Loading Data Using Firebase

In this chapter, we will cover the following tasks related to persistent data operations using Firebase:

- ▶ Saving array data to Firebase
- ▶ Rendering a large Firebase data set using collection-repeat
- ▶ Saving form data to Firebase

### Introduction

You will learn how to send and receive data between Ionic and a backend server, which is Firebase in our case. As an app developer, you will want to spend more time on building a solution for your customers than building a backend server with a database on your own. Firebase can act as a real-time database so that your app data can be synchronized between the database, frontend model objects, and the view layer (that is, what users actually see). This three-way binding is powerful because it simplifies many complex implementation scenarios such as the following:

- ▶ Chatting or messaging when there are multiple users sending and receiving data
- ▶ Saving a multistep form's data
- ▶ A real-time collaboration app that requires you to save a complex dataset
- ▶ A real-time data feed for a social networking app

The key benefit of Firebase as a backend database is that you can ensure that there is a very low latency between saving data to the server and receiving an acknowledgement. This is very critical in mobile app development, where the response time is critical to ensure a good user experience.

All the examples in this chapter will use AngularFire because it already has all the API operations provided for objects and arrays such as `add`, `remove`, `save`, and `get`. More information about the AngularFire API can be found at <https://www.firebaseio.com/docs/web/libraries/angular/api.html>.

If you have used AngularFire before the release of version 1.0, there have been some changes in the API implementation. Firebase has made the migration guide available at <https://www.firebaseio.com/docs/web/libraries/angular/migration-guides.html>.

## Saving array data to Firebase

Sending and saving data to the Firebase backend server is as simple as calling the `$add()` function. Firebase automatically handles all the REST operations so that you don't have to call via the `$http` services or create your own. From a frontend developer's perspective, you can treat your data as a typical JavaScript object.

In this recipe, you will learn how to make connections to Firebase by creating a reference object and adding data into the object in the array format. Firebase treats an array like an object. The only difference is that each array's item will have an integer key. Here's an example:

```
// When we send this
['a', 'b', 'c', 'd', 'e']
// Firebase stores this
{0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e'}
```

### Getting ready

You should log in to your Firebase dashboard ahead of time and navigate to the **Data** tab. The code will create a node called `items` to store data. You need to make sure that there is no existing `items` node, in case you created one before.

### How to do it...

Here are the instructions for adding 100 items in the `items` array stored in Firebase:

1. Create a blank Ionic app (such as `firebase-ionic`) and change the directory to that folder:

```
$ ionic start firebase-ionic blank
```

2. Edit `index.html` to make sure that Firebase and AngularFire (below `ionic.bundle.js`) are included:

```
<script src="lib/firebase.firebaseio-v2.2.4.js"></script>
<script src="lib/firebase/
angularfire-v1.1.1.min.js"></script>
```



You may wish to confirm with Firebase documentation for the latest versions and compatibility between Firebase and AngularFire. This example uses Firebase v2.2.4 and AngularFire v1.1.1 because they were the latest releases.

4. Open `app.js` to edit it. You may want to delete its entire contents because you will rewrite most of it.

5. Create two variables: `app` for the starter app and `ref` for the Firebase reference:

```
var app = angular.module('starter', ['ionic', 'firebase']);
var ref = new
Firebase('https://ionicebook.firebaseio.com/items/');
```



The `items` node does not need to exist in your Firebase application prior to creating a JavaScript reference to it. Firebase will automatically create it in the backend. If you want to create any node to store data, all you need to do is append the name in the URL. You should replace `ionicebook` with your own Firebase account.

6. Create the `app.run()` function as follows:

```
app.run(function($ionicPlatform, $firebaseArray,
$timeout) {
  $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();
    }
  });

  // $firebaseArray is new in 1.0.0
  // Instructions to migrate from previous version:
  // https://www.firebaseio.com/docs/web/
  // libraries/angular/migration-guides.html
  var items = $firebaseArray(ref);

  // Wait for items array to load from server
  items.$loaded().then(function(data) {
```

```
console.log('Number of items = ' + data.length);

// If there is no data, then add 100 items to the array
if (data.length == 0) {
    for (var i=0; i<100; i++) {
        data.$add({
            name: "Item " + i,
            $priority: i // Priority is used to ensure
            they are displayed in order
        });
    }
}
});
```

7. Go to the command line and run the app using `ionic serve`.
8. Navigate back to the **Data** tab of Firebase.
9. You should be able to see 100 records inserted in the `items` node, as shown in the following screenshot:



## How it works...

So far, you have not done anything in the Ionic interface. The preceding code only inserts 100 records in the `items` node in Firebase. This operation must happen within `items.$loaded().then()` because you have to ensure that Firebase finished its initialization of the connection. Then, for each record, you loop through a simple for loop and call `$add()`. The code also adds `$priority` in the JSON object. This is optional, because you want the data to be displayed properly in order later on. You can have any key and value in this JSON object.

## Rendering a large Firebase data set using collection-repeat

When your app grows, the size of the database also grows. Typically, there are many ways to handle large datasets such as paging, caching, and filtering. However, there is a very useful feature in Ionic that allows you to repeat through a large list of items without sacrificing performance. Ionic's `collection-repeat` is similar to AngularJS' `ng-repeat`. The main difference is its rendering mechanism. Most of the time, the app does not need to render thousands of items at once. So, `collection-repeat` accesses only a subset of the data and binds it to the current page, where it is visible to the user.

### Getting ready

You will continue to work on the previous app. There are some modifications in `index.html` and `app.js`.

### How to do it...

Here is how you can retrieve data and add, edit, and delete items:

1. Open `index.html` to edit it.
2. Insert the following code within `<body>`:

```
<ion-pane ng-controller="ItemCtrl">
  <ion-header-bar class="bar-calm item-input-inset">
    <label class="item-input-wrapper">
      <i class="icon ion-plus placeholder-icon"></i>
      <input type="text" placeholder="Insert New Item"
        ng-model="newItem">
    </label>
    <button class="button button-positive"
      ng-click="add(newItem); newItem='';">
      Add
    </button>
  </ion-header-bar>
  <ion-content>
    <ion-list>
      <ion-item class="item item-avatar"
        collection-repeat="item in items">
        
        <h2>{{ item.name }}</h2>
        <p>Description for {{ item.name }}</p>
    
```

```
<ion-option-button class="button-dark"
ng-click="edit(item)">
    Edit
</ion-option-button>
<ion-option-button class="button-assertive"
ng-click="delete(item)">
    Delete
</ion-option-button>
</ion-item>
</ion-list>
</ion-content>
</ion-pane>
```

You will write some code in `app.js` to handle `ItemCtrl` later. The preceding code is very simple, as it will perform the following functions:

- ❑ It can render all the items in a list.
  - ❑ If you swipe left, it shows the **Edit** and **Delete** buttons. This can be done via `<ion-option-button>`. You should always use it within the `<ion-list>` and `<ion-item>` hierarchy because that is how Ionic structures the directives.
  - ❑ The top header bar will have an input box and an **Add** button to add new records.
3. When a user edits an entry, there will be a modal popup with an input box to change the value of that entry. You need a template for the content of this Ionic modal. So, insert the following code after `<ion-pane>`:

```
<script type="text/ng-template" id="edit.html">
<ion-modal-view>
    <ion-header-bar>
        <h1 class="title">Edit Item</h1>
    </ion-header-bar>
    <ion-content>
        <div class="list">
            <label class="item item-input">
                <span class="input-label">Name</span>
                <input type="text" ng-model="editedItem.name">
            </label>
        </div>
        <button class="button button-full
button-positive" ng-click="save()">
            Save
        </button>
        <button class="button button-full
button-stable" ng-click="close()">
```

```
        Cancel
    </button>
</ion-content>
</ion-modal-view>
</script>
```

You can place the template code anywhere, either inside a `<script>` tag or in a separate HTML file, which needs to be linked either via a router or an Ionic modal delegate. This modal has *wired* the `editedItem.name` model to the input box. When a user clicks on the **Save** button, it will call the `save()` function to send data to Firebase.

4. Now, let's work on `app.js` to create the logic in your `ItemCtrl`:

```
app.controller('ItemCtrl', function($scope,
$firebaseArray, $ionicModal, $ionicListDelegate) {

$scope.items = $firebaseArray(ref);
$scope.editedItem = {};

$scope.items.$loaded().then(function(data) {
// Start the priority value which is lower
// than previous
// The lower the value, the higher it is
// appearing on the list
$scope.newPriority = (data.length > 0)
? data[0].$priority - 1 : -1;
console.log("Items have been loaded:");
console.log(data);
});

// Load template for modal to edit an item
$ionicModal.fromTemplateUrl('edit.html', {
scope: $scope,
animation: 'slide-in-up'
}).then(function(modal) {
$scope.modal = modal;
});
});
```

You need to load the Firebase data again to `$scope.items` so that the view has access to the array. After the data is loaded, `$scope.newPriority` is assigned with a value that is smaller than the latest priority value in the zero index of the array. The `$ionicModal.fromTemplateUrl()` function is also called to load the template in the memory. Then, the entire modal directive is assigned to the `$scope.modal` object. This will allow you to programmatically show and hide the modal later.

5. Also, within `ItemCtrl`, you need to add those functions to handle interactions from the view layer:

```
$scope.add = function(val) {
  if (!val || val.length === 0 || !val.trim()) {
    alert("Your value is invalid");
  } else {
    $scope.items.$add({
      name: val,
      $priority: $scope.newPriority
    });
    $scope.newPriority--;
  }
};

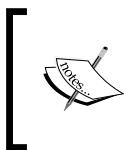
$scope.edit = function(item) {
  $scope.editedItem = item;
  $scope.modal.show();
};

$scope.save = function() {
  $scope.items.$save($scope.editedItem);
  $scope.modal.hide();
  $ionicListDelegate.closeOptionButtons();
};

$scope.close = function() {
  $scope.modal.hide();
  $ionicListDelegate.closeOptionButtons();
};

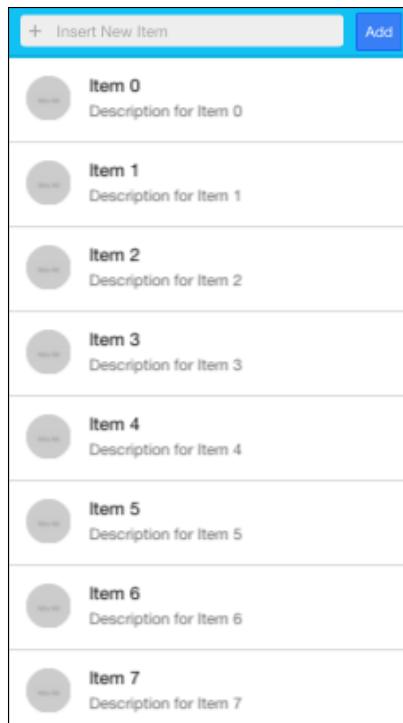
$scope.delete = function(item) {
  $scope.items.$remove(item);
  $ionicListDelegate.closeOptionButtons();
};
```

Deleting a record is very simple; you just have to call `$scope.items.$remove()`. You can pass an index of the record or the record itself, which is an object reference. You can add a record by calling `$scope.items.$add()` and passing the record content. AngularFire will take care of the entire REST API operation. The `edit()` function here is actually used to just show the modal and assign the item object to `$scope.editedItem`.



The reason behind this assignment is that the modal needs to know which record it is dealing with. So, it's best to make a separate reference and have it available for the modal. Once the user edits the content of this record, `$scope.items.$save()` will take care of the save operation.

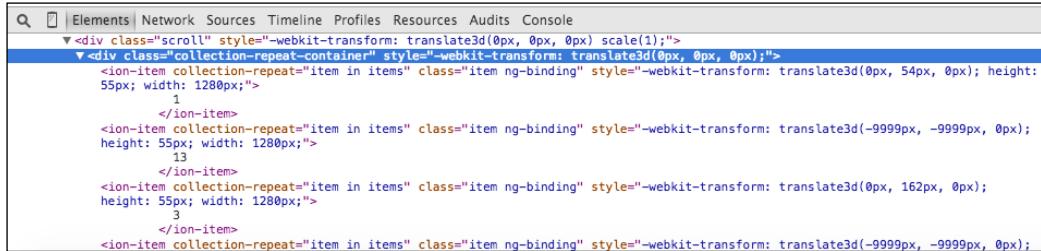
6. Go back to the command line to run the app via `ionic serve`.
7. You should be able to see the following in the browser:



8. Open the browser console to take a look at the array object's output. This is just done so that you understand what is returned to the browser.

 A screenshot of a browser's developer tools console. The title bar says "Console". The console output shows a GET request to "http://localhost:8080/cordova.js" and the response, which is a large JSON object representing an array of items. The array has 21 elements, with the first few items shown in red. The first item is an object with properties: "id": "-Juoz5\_3-zhamaGqubP", "priority": 0, "name": "Item 0", and a long string for the description. The "Object" keyword is repeated many times in the output. The right side of the console shows the URL "localhost:121" and the ports "app.15147" and "app.15148".

9. Navigate to the **Elements** tab of the console. If you scroll down, note that the `<ion-item>` content is actually being updated dynamically. This is the result of using `collection-repeat`, as it only keeps one page of records in the DOM. When you scroll down, `collection-repeat` will update the `<ion-item>` tag instead. This allows infinite scrolling without you realizing that you are going through the content page by page.



The screenshot shows the browser's developer tools with the "Elements" tab selected. It displays the DOM structure of a list generated by the `collection-repeat` directive. The root element is a `<div>` with a class of "scroll". Inside it is another `<div>` with a class of "collection-repeat-container". This container holds multiple `<ion-item>` elements, each with a class of "item ng-binding". The items are styled with a height of 55px and a width of 1280px. The list is scrollable, as indicated by the "scroll" class and the translate3d CSS transform applied to the container.

## How it works...

The app created in the preceding recipe can be very complicated if you build your own backend and frontend code to achieve the same scrolling performance. The key here is to assign `$firebaseArray()` to some object within `$scope` so that you can perform data operations while allowing the function to bind data at the view level. There is no need to create another `ItemViewModel` instance, especially for situations such as when you need to just handle the synchronization between frontend data and the view model.

It's best to use `collection-repeat` whenever possible in your Ionic app. The data in `collection-repeat` must be an array, which is similar to how `ng-repeat` works. You cannot use one-time binding (`::model`), because `collection-repeat` needs to update the model for each item in the list. You don't need to worry about the number of items in the list as well as the height of the item on the screen. Also, it's a good practice to cache images if `<img>` is used for `collection-repeat`.

## There's more...

For more information about `collection-repeat`, visit <http://ionicframework.com/docs/api/directive/collectionRepeat/>.

## Saving form data to Firebase

This recipe will cover another example of using Firebase as a persistent storage layer for your app. A very common use case that you will see in every app is the ability to have users fill out a form and save data to the server. Sometimes, there is more than one form in the entire process. A good example is a multistep shopping cart checkout app.

You will create an app that will:

- ▶ Have three screens with a form per screen
- ▶ Allow a user to either swipe or click on the **Back** and **Next** buttons to navigate horizontally to each screen
- ▶ Let a user submit the form and save it to Firebase

## Getting ready

You can close the previous app from your editor because this example will start with a brand new app.

## How to do it...

Here are the steps:

1. Let's start with a blank Ionic app (such as `firebase-ionic-form`) and include `firebase-v2.2.4.js` and `angularfire-v1.1.1.min.js` in the `index.html` header, as follows:

```
$ ionic start firebase-ionic-form blank
```

2. Structure the `index.html` file in such a way that it has a header and a body area with a slide box. But first, you need to assign a controller and a full-screen class so that the form can take the entire space of the device screen:

```
<body ng-app="starter" ng-controller="FormCtrl"
      class="full-screen">
  Open /www/css/style.css to add some custom styles
  .full-screen {
    position: fixed;
  }

  body > div.has-header.full-screen {
    height: 100%;
  }

  .slider {
    height: 100%;
  }
```

By default, Ionic's slider will only expand as far as the inner content goes. So, if your content only takes half of the space, Ionic's slider won't expand all the way to the bottom. This means that users cannot swipe if they touch the second half of the screen, because Ionic's slider cannot detect a touch event outside its own covered area.

3. Add the header within the `<body>` tag, as follows:

```
<div class="bar bar-header">
  <button class="button" ng-if="data.currentSlide > 0"
  ng-click="data.currentSlide = data.currentSlide - 1">
    <i class="icon ion-chevron-left"></i>
    Back
  </button>
  <h1 class="title">Step {{ data.currentSlide + 1 }}</h1>
  <button class="button button-positive"
  ng-if="data.currentSlide < 2" ng-click="data.currentSlide
  = data.currentSlide + 1">
    Next
    <i class="icon ion-chevron-right"></i>
  </button>
  <button class="button button-positive"
  ng-if="data.currentSlide == 2" ng-disabled="data.
  completed" ng-click="submit()">
    Purchase
  </button>
</div>
```

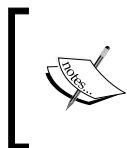
There is a lot going on in this header. But for the most part, the user will see a **Back** and **Next** button. The **Purchase** button only appears when you are on the last slide (which is slide 2) of the slider. The slide box will bind to this `data.currentSlide` variable so that you can get the current value or assign a value to it in order to programmatically change the slide.

4. Insert the following `ion-slide-box` code:

```
<div class="has-header full-screen">
  <ion-slide-box active-slide="data.currentSlide">
    <ion-slide>
      <div class="list">
        <label class="item item-input">
          <span class="input-label">First</span>
          <input type="text" ng-model="formData.first">
        </label>
        <label class="item item-input">
          <span class="input-label">Last</span>
          <input type="text" ng-model="formData.last">
        </label>
        <label class="item item-input">
          <span class="input-label">Email</span>
          <input type="email" ng-model="formData.email">
        </label>
      </div>
    </ion-slide>
  </ion-slide-box>
</div>
```

```
<ion-slide>
  <div class="list">
    <label class="item item-input">
      <input type="text" pattern="\d*"
        placeholder="Card Number" ng-model="formData.cc">
    </label>
    <label class="item item-input">
      <input type="text" pattern="\d*" placeholder
        ="CVC" maxlength="4" ng-model="formData.cvc">
    </label>
    <label class="item item-input">
      <input type="text" pattern="\d*" placeholder
        ="Expiration" maxlength="4"
        ng-model="formData.exp">
    </label>
  </div>
</ion-slide>
<ion-slide>
  <ul class="list">
    <li class="item item-input item-select">
      <div class="input-label">
        LCD TV
      </div>
      <select ng-model="formData.model">
        <option value="60-in" selected>60
        Inches</option>
        <option value="65-in">65 Inches</option>
        <option value="70-in">70 Inches</option>
      </select>
    </li>
    <li class="item item-toggle">
      Extended Warranty
      <label class="toggle toggle-assertive">
        <input type="checkbox"
          ng-model="formData.warranty">
        <div class="track">
          <div class="handle"></div>
        </div>
      </label>
    </li>
  </ul>
  <h4 class="balanced" ng-if="data.completed">Your
  purchase has been completed!</h4>
</ion-slide>
</ion-slide-box>
</div>
```

You must declare `active-slide="data.currentSlide"` in order for the slide box directive to know the variable on which to perform two-way binding. The structure of a slide box is similar to `<select>` and `<option>` in HTML. You basically start the group with `<ion-slide-box>`, and for each box, you just need to put the HTML content inside `<ion-slide>`.



The rest of the code comprises just regular form elements such as select, text, and number inputs. Bind each input with a model such as `formData.first` (that is, first name) or `formData.email` (that is, email).



5. Open `app.js` to edit it. You may want to delete the entire content and insert the following in it:

```
var app = angular.module('starter', ['ionic', 'firebase']);

app.run(function($ionicPlatform, $firebaseArray,
$timeout) {
    $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();
        }
    });
});

app.controller('FormCtrl', function($scope,
$firebaseArray) {
    var ref = new Firebase('https://ionicebook.firebaseio
    .com/transactions/');
    $scope.transactions = $firebaseArray(ref);
    $scope.formData = {};
    $scope.data = {
        completed: false,
        currentSlide: 0
    }
});
```

```

};

// If user changes slide, make notification disappear
$scope.$watch('data.currentSlide',
function(newVal, oldValue) {
    if ((newVal < 2) && (oldVal == 2))
        $scope.data.completed = false;
})

$scope.submit = function() {
    // Check if the form is "dirty" or not as user
    // must fill out something
    if (angular.equals({}, $scope.formData)) {
        alert("Your form is empty");
    } else {
        // Add the entire $scope.formData object to
        // Firebase and reset it
        $scope.transactions.$add($scope.formData) .
        then(function(res) {
            $scope.formData = {};
            $scope.data.completed = true;
            // Mark "completed" to show the notification
            // in view
        });
    }
};
}
);

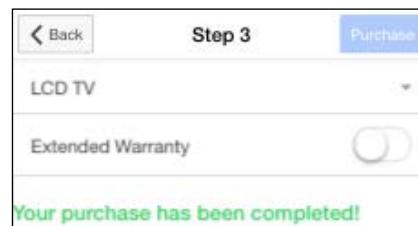
```

Your FormCtrl controller is very simple in this case. In a real-world application, there are more steps to validate the form data. For example, the customer name must be alphabetical in nature and a credit card's expiration date must be valid. However, this section will not delve into validation. The focus here is to explain how to just send *data* to Firebase by calling the `$scope.transactions.$add()` function.

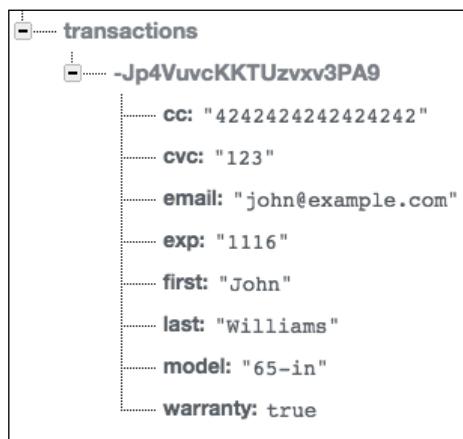
6. Run the app and fill out some fake data in the form, as follows:

Step 1	
First	John
Last	Williams
Email	john@example.com

7. Click on **Purchase**, and you will see the **Your purchase has been completed!** confirmation text. This means that the app has successfully sent the data to Firebase and all the form models are reset:



8. To confirm that the data existed in Firebase, go to your Firebase app and look at the transactions node. You will see a JSON, like the following screenshot:



This means that all the form data has been saved successfully to your Firebase app.

## How it works...

The simple example in the preceding section helps demonstrate a few key concepts. These concepts are as follows:

- ▶ You can have multiple forms using any method such as a slide box or another route in a Single-Page Application structure. You can save data to Firebase in the last step using one simple call to `$add()`.
- ▶ By default, Ionic's slide box does not cover the entire screen. You may want to modify some styles to change this behavior.
- ▶ Form data is just a JSON model. You can bind each value to an input box.

- ▶ The `active-slide` option of `<ion-slide-box>` is the two-way data binding. So, you can get the current slide or change it to make the slide box jump to a particular slide.
- ▶ There are many ways that can be used to detect the state of your form: `empty`, `valid`, `invalid`, `submitted`, and so on. The preceding example does not go deep into validation but just provides a few easy paths to check the state using the `data.currentSlide` or `data.completed` variables.

In addition to this, note the use of the `$scope.$watch('data.currentSlide', function(newVal, oldVal)` in the controller. Since the app provides users with two ways to navigate each slide (either click on the **Back** and **Next** button, or swipe), you need to actually bind data to the live value of the slide number. So, if the user navigates away from the last slide (which is slide 2), you basically tell the view to hide the notification text. By default, the text does not appear until after the `submit()` function is called.



# 10

## Finalizing Your Apps for Different Platforms

In this chapter, we will cover the following tasks related to building and publishing apps:

- ▶ Building and publishing an app for iOS
- ▶ Building and publishing an app for Android
- ▶ Using PhoneGap Build for cross-platform applications

### Introduction

In the past, it used to be very cumbersome to build and successfully publish an app. However, there is much documentation and many unofficial instructions on the Internet today that can pretty much address any problem that you may run into. In addition, Ionic also comes with its own CLI to assist in this process. This chapter will guide you through the app building and publishing steps at a high level. You will learn how to:

- ▶ Build an iOS and Android app via the Ionic CLI
- ▶ Publish an iOS app using Xcode via iTunes Connect
- ▶ Build a Windows Phone app using PhoneGap Build

The purpose of this chapter is to provide ideas on what to look for and some *gotchas*. Apple, Google, and Microsoft are constantly updating their platforms and processes. So, the steps may not look exactly the same over time.

## Building and publishing an app for iOS

Publishing on the App Store can be a frustrating process if you are not well prepared upfront. In this recipe, we will go through the steps that are required to properly configure everything in the Apple Developer Center, iTunes Connect, and a local Xcode project.

### Getting ready

You must register for the Apple Developer Program in order to access <https://developer.apple.com> and <https://itunesconnect.apple.com> because these websites will require an approved account.

In addition to this, the instructions in the subsequent recipes use the latest version of the following components:

- ▶ Mac OS X Yosemite 10.10.4
- ▶ Xcode 6.4
- ▶ Ionic CLI 1.6.4
- ▶ Cordova 5.1.1

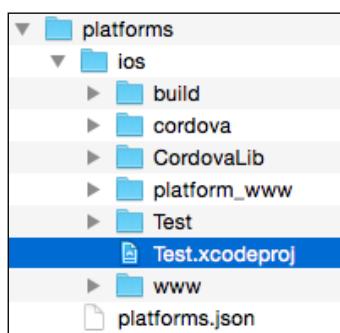
### How to do it...

Here are the instructions to build and publish an app for iOS:

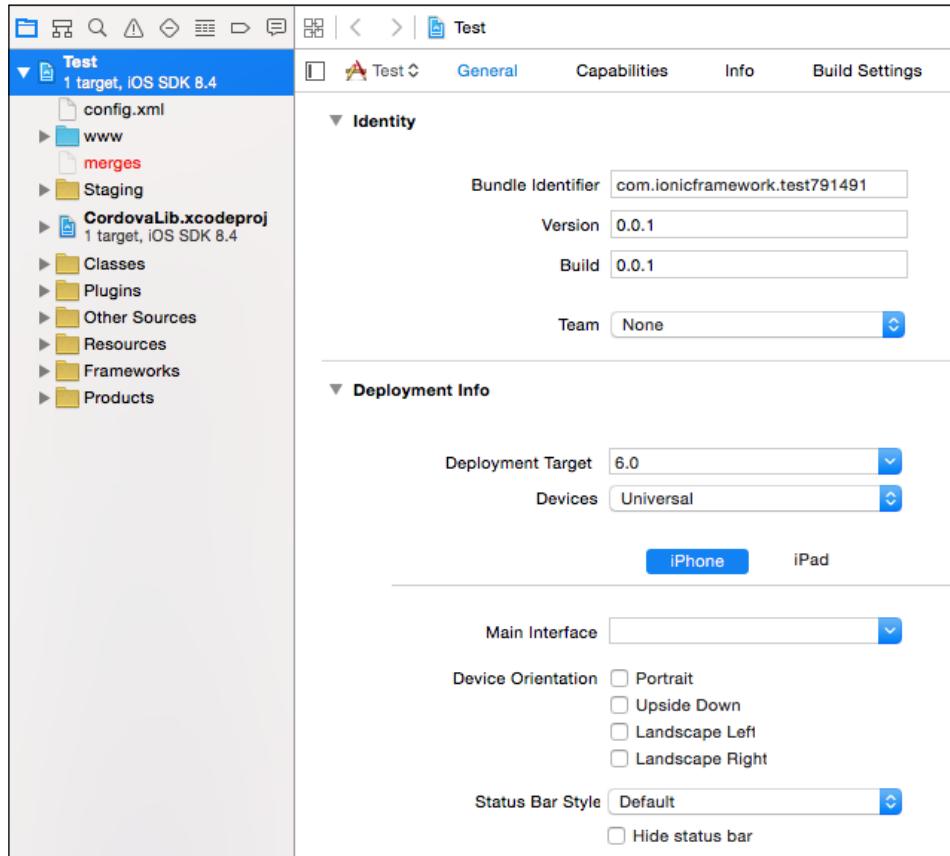
1. Make sure that you are in the app folder and build for the iOS platform:

```
$ ionic build ios
```

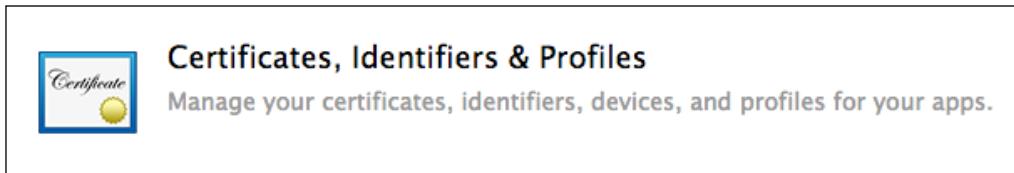
2. Go to the `ios` folder /`platforms` and open the `.xcodeproj` file in Xcode:



3. Go through the **General** tab to make sure that you have the correct information for everything, especially the information related to Bundle Identifier and version. Change and save this as needed:



4. Visit the Apple Developer website and click on **Certificates, Identifiers & Profiles**:



5. For an iOS app, you just have to go through the steps on the website and fill out the necessary information:

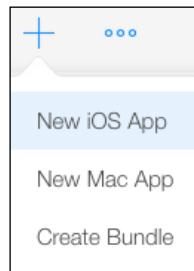


The important part that you need to do correctly here is, going to **Identifiers | App IDs** because it must match your Bundle Identifier in Xcode.

6. Visit iTunes Connect and select the **My Apps** button:



7. Select the Plus (+) icon and click on **New iOS App**:



- Fill out the form and make sure that you select the right Bundle Identifier for your app:

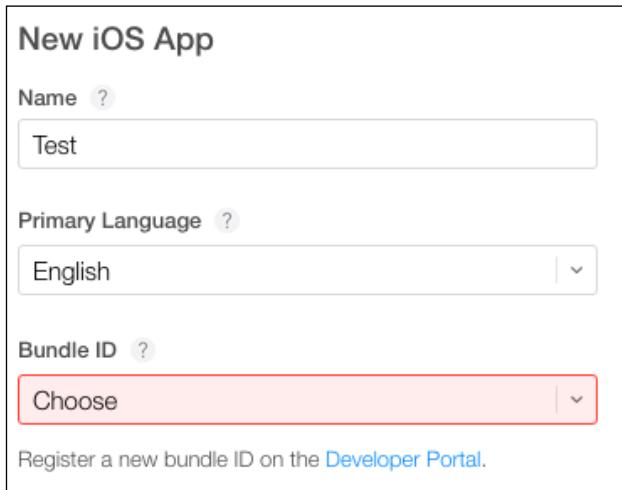
New iOS App

Name ?  
Test

Primary Language ?  
English

Bundle ID ?  
Choose

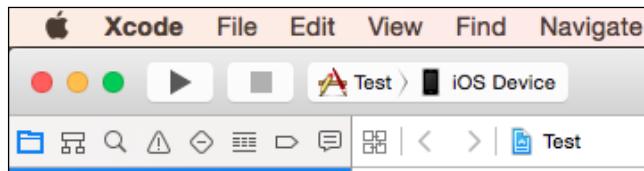
Register a new bundle ID on the [Developer Portal](#).



There are several additional steps that require you to provide information about the app such as screenshot, icon, address, and so on. If you just want to test the app, you can just provide some placeholder information initially and come back to edit it later.

That's it when it comes to preparing your Developer and iTunes Connect account.

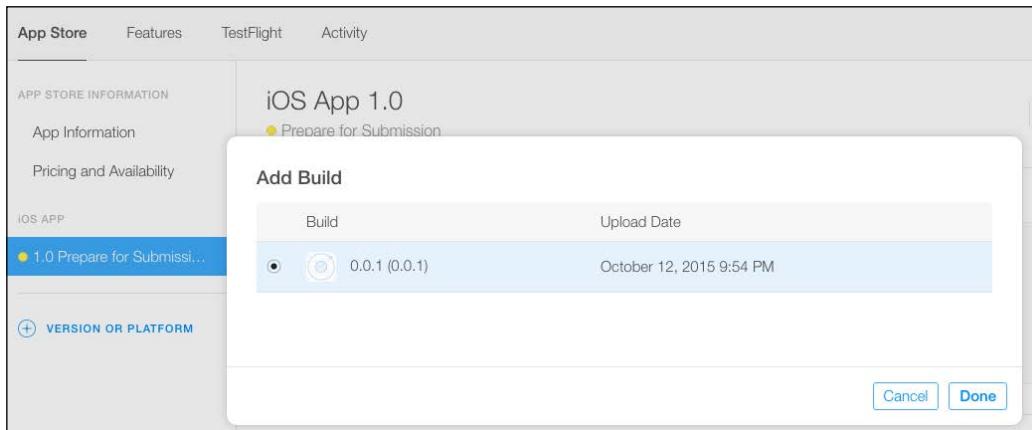
- Now, open Xcode and select **iOS Device** as the archive target. Otherwise, the Archive feature will not be turned on. You will need to archive your app before you can submit it to the App Store.



10. Navigate to **Product | Archive** in the menu at the top:



11. After the archive process is completed, select **Submit to App Store** to finish the publishing process.
12. At first, the app may take an hour to appear in iTunes Connect. However, subsequent submissions will happen faster. You should look for the build in the **iOS App 1.0 Prepare for Submission** menu in iTunes Connect. Then select the build version (which is 0.0.1 in this case) and click on the **Done** button.:.



iTunes Connect has a very nice integration with TestFlight to test your app. You can switch this feature on and off. Note that for each publication, you have to change the version number in Xcode so that it won't conflict with the existing version in iTunes Connect.

13. To publish the app, select **Submit for Beta App Review**. You may want to go through other tabs such as Pricing and In-App Purchases to configure your own requirements.

## How it works...

This recipe obviously does not cover every bit of detail in the publishing process. In general, you just need to make sure that your app is locally tested thoroughly in a physical device (either via USB or TestFlight) before submitting it to the App Store.

If for some reason the Archive feature doesn't build, you can manually go to the local Xcode folder to delete the specific temporarily archived app to clear the cache, as follows:

```
~/Library/Developer/Xcode/Archives
```

## See also

TestFlight is a separate subject in itself. The benefit of TestFlight is that you don't need your app to be approved by Apple in order to install the app on a physical device for internal testing and development. You can find out more about TestFlight at [https://developer.apple.com/library/prerelease/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect\\_Guide/Chapters/BetaTestingTheApp.html](https://developer.apple.com/library/prerelease/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/BetaTestingTheApp.html).

# Building and publishing an app for Android

Building and publishing an Android app is a little more straightforward than iOS because you just interface with the command line to build an .apk file and upload it to Google Play Developer Console. The Ionic Framework documentation also has a great instruction page for this, which can be viewed by visiting <http://ionicframework.com/docs/guide/publishing.html>.

## Getting ready

The requirement is to have your Google Developer account ready and log in to <https://play.google.com/apps/publish>. Your local environment should also have the right SDK as well as the keytool, jarsigner, and zipalign command line for that specific version.

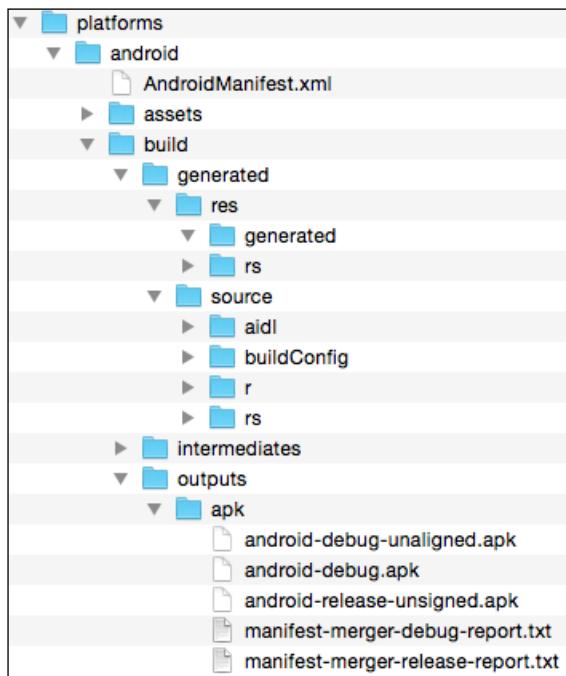
## How to do it...

Here are the instructions to build and publish an app for Android:

1. Go to your app folder and build the release for Android:

```
$ ionic build --release android
```

2. You will see android-release-unsigned.apk in the apk folder under / platforms/android/build/outputs. Go to this folder in the Terminal:



3. If you're creating this app for the first time, you must have a keystore file. This file is used to identify your app for publishing. If you lose it, you cannot update your app later on. To create a keystore, type the following command line and make sure that it's the same keytool version of the SDK:

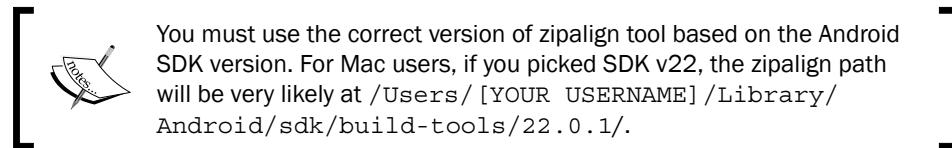
```
$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

4. Once you fill out the information in the command line, make a copy of this file and keep it somewhere safe because you will need it later.
5. The next step is to use this file to sign your app so that it will create a new .apk file that Google Play allows users to install:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore HelloWorld-release-unsigned.apk alias_name
```

6. To prepare for the final .apk file before upload, you must package it using zipalign, as follows:

```
$ zipalign -v 4 HelloWorld-release-unsigned.apk HelloWorld.apk
```



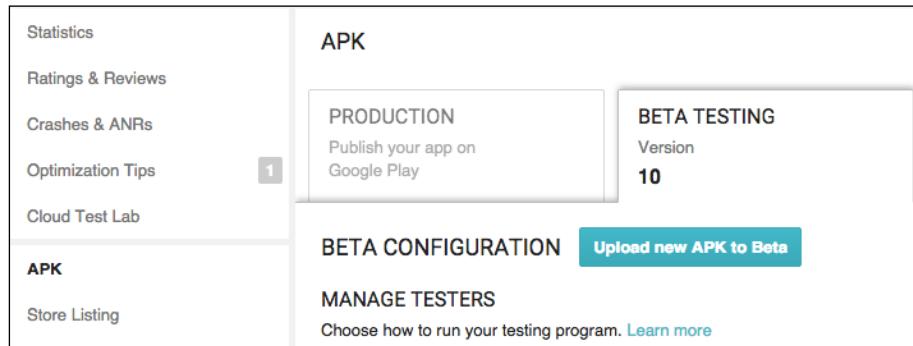
7. Log in to the Google Developer Console and select **Add new application**:



8. Fill out as much information for your app as possible using the menu to the left:

The screenshot shows the 'Store Listing' section of the Google Play Developer Console. On the left, there is a sidebar with various tabs: Statistics, Ratings & Reviews, Crashes & ANRs, Optimization Tips (with a '1' notification), Cloud Test Lab, APK, Store Listing (which is selected and highlighted in blue), Experiments, Content Rating, Pricing & Distribution, In-app Products, and Services & APIs. The main right-hand panel is titled 'STORE LISTING' and contains the 'PRODUCT DETAILS' section. Under 'PRODUCT DETAILS', there is a dropdown menu set to 'English (United States) – en-US'. Below the dropdown, there are three input fields: 'Title\*' with the value 'English (United States) – en-US', 'Short description\*' with the value 'English (United States) – en-US', and 'Full description\*' with the value 'English (United States) – en-US'.

- Now, you are ready to upload the .apk file. First perform a Beta test:



- Once you are done with Beta testing, you can follow the Developer Console instructions to push the app to **Production**.

## How it works...

This recipe does not cover other Android marketplaces such as Amazon Appstore because each of them has different processes. However, the common idea is that you need to completely build the unsigned version of .apk, sign it using an existing or new keystore file, and finally zipalign to prepare for upload.

## Using PhoneGap Build for cross-platform applications

Adobe PhoneGap Build is a very useful product that provides *build-as-a-service* in the cloud. If you are having trouble building an app locally in your computer, you can upload the entire Ionic project to PhoneGap Build, and it will build the app for Apple, Android, and Windows Phone automatically.

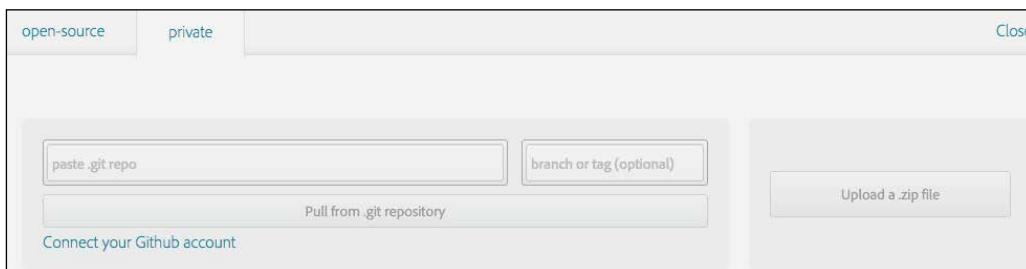
## Getting ready

Go to <https://build.phonegap.com> and register for a free account. You will be able to build one private app for free. For additional private apps, there is a monthly fee associated with the account.

## How to do it...

Here are the instructions to use PhoneGap Build for cross-platform applications:

1. Zip the entire /www folder and replace cordova.js with phonegap.js in index.html, as described in [http://docs.build.phonegap.com/en\\_US/introduction\\_getting\\_started.md.html#Getting%20Started%20with%20Build](http://docs.build.phonegap.com/en_US/introduction_getting_started.md.html#Getting%20Started%20with%20Build).
2. You may have to edit config.xml to ensure that all the plugins are included. Detailed changes are available in PhoneGap documentation, which can be viewed by visiting [http://docs.build.phonegap.com/en\\_US/configuring\\_plugins.md.html#Plugins](http://docs.build.phonegap.com/en_US/configuring_plugins.md.html#Plugins).
3. Select Upload a .zip file under the **private** tab:



4. Upload the ZIP file of the www folder.
5. Make sure that you upload an appropriate key for each platform. For a Windows Phone, upload the publisher ID file.
6. After this, you just build the app and download the completed build file for each platform.

## How it works...

In a nutshell, PhoneGap Build is a convenient way of building apps when you are only familiar with one platform during the development process but you want your app to be built quickly for other platforms. Under the hood, PhoneGap Build has its own environment to automate the process for each user. However, the user still has to own the responsibility of providing a key file to sign the app. PhoneGap Build just helps you attach the key to your app.

## See also

A common issue that people face when using PhoneGap Build is a failure to build. You may want to refer to their documentation to troubleshoot. This documentation is available at [http://docs.build.phonegap.com/en\\_US/support\\_failed-builds.md.html#Failed%20Builds](http://docs.build.phonegap.com/en_US/support_failed-builds.md.html#Failed%20Builds).

# Bibliography

This Learning Path is a blend of content, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Getting Started with Ionic, Rahat Khanna*
- *Ionic Framework By Example, Sani Yusuf*
- *Ionic 2 Cookbook, Second Edition, Hoc Phan*



# Index

## Symbols

\$http constructor method 87, 88  
\$http services  
  \$http constructor method 87, 88  
  about 86, 87  
  response object 87  
\$ionicBackdrop 75  
\$ionicConfigProvider  
  about 83, 221, 233  
  URL 233  
\$ionicGesture service  
  about 76  
  off method 77  
  on method 76  
\$ionicLoading 80  
\$ionicModal  
  about 81  
  IonicModal controller 81  
\$ionicPopover 82  
\$ionicPopover service  
  adding 211  
  code, finishing 212  
\$ionicPosition 82  
\$resource  
  and REST API 92  
\$scope.\$digest() function 366  
<ion-nav-bar> directive 61  
<ion-nav-title> directive 61  
<ion-side-menu-content> directive 60  
<ion-side-menu> directive 59  
<ion-side-menus> directive  
  about 59  
  delegate-handle(optional) attribute 59  
  enable-menu-with-back-views(optional)  
    attribute 59

<ion-tab> directive  
  about 58  
  badge (optional) 58  
  href (optional) 58  
  icon-off (optional) 58  
  icon-on (optional) 58  
  icon (optional) 58  
  on-deselect (optional) 59  
  on-select (optional) 58  
  title attribute 58  
<ion-tab> element 241, 242  
<ion-tabs> directive  
  about 57  
  delegate-handle(optional) attribute 58  
  tabs-icon-left 57  
  tabs-icon-only 57  
  tabs-icon-top 57  
  tabs-<theme\_name> 57  
<ion-view> directive  
  back menus 61

## A

abstract state 240  
account, Firebase  
  setting up 255-258  
addUser() function 391  
anatomy, Hybrid Mobile Application  
  about 6  
  custom WebView 7  
  native code, calling to JS Bridge 8  
  native library 7  
Android 141  
Android device  
  todo app, running 170

**Angular code, Bucket-List app**  
Add button, wiring up 186  
controller, binding 186  
input box model, binding 186  
ion-item, binding 187  
text, entering into input box 182  
wring, to UI 186  
writing 182

**AngularFire API**

reference 484

**AngularJS**

about 8  
concepts 9  
controllers 10, 279  
directive 10, 279  
expressions 13  
filter 280  
filters 13  
modules 9  
service 11, 12, 280  
templates 12  
URL 271

**Angular UI Router**

about 48  
abstract state 51  
dot notation, using 49  
multiple views, setting 51  
nested states 49  
nested views, views 50  
object-based states, using 50  
parent property, using 50  
state parameters 52  
states 48, 49  
transition, to state 50  
URLs 48, 49  
views 49

**Apache Cordova**

about 143, 144  
using 13

**app**

animating, requestAnimationFrame using  
with event binding 434-441  
building, for Android 507-510  
building, for iOS 502-506  
publishing, for Android 507-510  
publishing, for iOS 502-506

**Appcamp.IO 265**

**app.js file 38**

**Apple Push Notification Services (APNS) 117, 132**

**array data**

saving, to Firebase 484-486

**attributes, ion-content directive**

direction [string] 39  
has-bouncing [boolean] 39  
overflow-scroll [boolean] 39  
scroll [boolean] 39  
start-x [string] 39  
start-y [string] 39

## B

**blank template**

about 40  
Ionic, setting up with 279

**BooksFactory 90, 91**

**BookStore**  
about 44  
architecture 44  
design 44  
features 44  
layout 61  
navigation 61

**Bower**

URL 271

**Brackets**

URL 27

**Bucket-List app**

about 173  
Angular code, writing 182  
breaking down 174  
creating 173, 174  
overview 174  
testing 189  
UI, designing 174

**buttons, CSS components**

about 65, 66  
bar 66  
icon buttons 66

## C

**camera plugin 114-116**

**capabilities, Ionic Lab**

app creation 136

app logs 136  
 app, previewing 136  
 app, running 136  
 builds, making 136  
**cards, CSS components** 68  
**CDN-hosted library files**  
 using 30, 31  
**Code Editors**  
 about 27  
 brackets 27  
 Sublime Text 28  
 Visual Studio 28  
**code pen, Ionic**  
 about 267  
 URL 267  
**communication, between various components**  
 controller to directive 400  
 directive to factory 400  
 enabling, events used 400-408  
 factory to factory 400  
 state to view and/or controller 400  
 view to controller 400  
 view to view 400  
**config method** 9  
**config.xml**  
 URL 36  
**contact**  
 adding 348-354  
 picking 347-355  
**contacts plugin** 121  
**Continuum** 168  
**controllers, AngularJS** 10, 11  
**Cordova**  
 URL 272, 331  
**Cordova Google Maps plugin**  
 features 367  
 reference 355  
**Cordova plugin**  
 about 112  
 custom development 127  
 managing 113  
**Cordova Video Player plugin**  
 URL 342  
**Cross compiled Hybrid Apps** 5  
**CSS3 filters**  
 URL 338

**CSS components**  
 about 64  
 buttons 65  
 cards 68  
 footer 65  
 forms 69  
 grid 72  
 header 64  
 input elements 69  
 lists 67  
 tabs 71  
 utility 73  
**custom filter**  
 creating 429-433

**D**

**data, London Tourist App (LTA)**  
 designing 201, 202  
 retrieving, with \$http service 202-205  
 wiring up 202  
**deleteBucketListItem function** 185  
**Dependency Injection (DI)** 9, 88  
**development environment**  
 setting up 276-278  
**device camera**  
 used, for capturing photo 332-337  
**directives, AngularJS** 10  
**documentation page, Ionic**  
 about 266  
 URL 266  
**drag events**  
 detecting, with gesture coordinate 394-399  
**dummy app**  
 building 25, 26

**E**

**email**  
 composing with attachment,  
     from app 342-346  
**events, JS components** 76  
**examples**  
 copying, from Ionic Codepen  
     Demos 285, 286  
**expressions, AngularJS** 13

## F

### **Facebook app**

configuring, with Firebase authentication 445-449

### **features, Ionic**

about 265  
Appcamp.IO 265  
code pen 267  
community 270  
CSS 146  
documentation page 266  
Ionic creator 267  
Ionic.IO 268  
JavaScript 147, 149  
playground 269

### **features, Ionic v2**

abstracted annotations 137  
Angular 2 136  
enhanced native integration 137  
ES6 136  
improved navigation and routing 137  
material design 137  
powerful theming 137  
TypeScript 136

### **files, Ionic Project**

bower.json 36  
config.xml 36  
gulpfile.js 36  
package.json 36

### **Filterous**

URL 337

### **filters, AngularJS 13**

### **Firebase**

about 255  
adding 258  
Angular-Fire scripts, adding 258-260  
array data, saving to 484-486  
data, adding to database 262  
data, pulling from database 260, 261  
form data, saving to 492-498  
implementing, into Ionic tabs  
    application 260  
integrating, into Ionic tabs  
    application 258-262  
integration 97

large dataset, rendering with collection-repeat 487-492

new account, setting up 255-258

pulling, from database 260

URL 97, 255, 257, 271

### **Firebase authentication**

for configuring Facebook app 445-449  
for configuring Google+ project 453-457  
for configuring Twitter app 450-453

### **folders, Ionic Project**

hooks 36  
platforms 36  
plugins 36  
resources 36  
scss 37  
www 37

### **footer, CSS components 65**

### **form inputs, JS components**

about 75  
ion-checkbox directive 75  
ion-radio directive 76  
ion-toggle directive 76

### **forms, CSS components 69**

## G

### **Genymotion for Android**

for viewing HelloWorld app 294-301

### **Geolocation Cordova plugin 119, 120**

### **gesture events**

about 77  
on-double-tap 77  
on-drag 77  
on-drag-down 78  
on-drag-left 78  
on-drag-right 77  
on-drag-up 77  
on-hold 77  
on-release 77  
on-swipe 78  
on-swipe-down 78  
on-swipe-left 78  
on-swipe-right 78  
on-swipe-up 78  
on-tap 77  
on-touch 77

**gestures, JS components** 76  
**getGroupsAll() function** 391  
**getGroupsByUserId() function** 391  
**git**  
    URL 26  
**Google account**  
    URL 255  
**Google Chrome**  
    about 164  
    emulation feature, using 165, 167  
    URL 164  
**Google Cloud Messaging (GCM)** 117, 132  
**Google Maps**  
    adding, with geocoding support 355-368  
**Google+ project**  
    configuring, with Firebase  
        authentication 453-457  
**gotoMyLocation() function** 366  
**Greensock**  
    URL 441  
**grid, CSS components** 72  
**Gulp**  
    URL 271

## H

**header, CSS components** 64  
**HelloWorld app**  
    creating, via CLI 278-280  
    creating, via Ionic Creator 281-284  
    folder structure, customizing 305  
    viewing, Genymotion for Android used 294-301  
    viewing, Ionic View used 301-304  
    viewing, iOS Simulator used 290  
    viewing, web browser used 287-290  
    viewing, Xcode for iOS used 291-293  
**hybrid applications**  
    about 143, 144  
    issues 144, 145  
**Hybrid Mobile Application**  
    about 3, 4  
    anatomy 6  
    types 5

## I

**index.html file** 37

**input box, Bucket-List app**  
    array, creating for Bucket-List items 184  
    code, implementing for Add button 184  
    controller, creating 182, 183  
    Delete button, implementing 185  
    model, creating 183  
    text, entering into 182  
**input elements, CSS components**  
    about 69  
    checkbox 70  
    radio button list 70  
    range control 71  
    toggle control 71  
**Instagram plugin**  
    URL 337  
**introduction screen**  
    creating, with custom header 413-421  
**ion-content** 39  
**ion-content directive**  
    attributes 39  
**ion-footer-bar directive**  
    about 57  
    align-title(optional) attribute 57  
**ion-header-bar directive**  
    align-title(optional) attribute 56  
    no-tap-scroll(optional) attribute 56  
**Ionic**  
    about 129, 145  
    advantages 276  
    features 146, 265  
    history 145  
    jobs, URL 272  
    lab, URL 272  
    resources 271  
    setting up 150, 151  
    showcase, URL 272  
    stack overflow, URL 271  
    standard templates 278  
**Ionic Analytics** 134  
**Ionic application**  
    blank template 153  
    creating 153, 154  
    customizing 221  
    ngCordova, integrating to 113, 114  
    side menu template 153  
    tabs template 153  
    to-do list app, creating 154

**ionic.app.scss file** 225, 226

**Ionic CDN**

URL 31

**Ionic CLI**

about 149, 150

features 149

installing 150, 151

reference 279

using, locally 34

**Ionic cloud services**

about 130

Ionic Analytics 134

Ionic Creator 130

Ionic Deploy 133

Ionic Lab 135

Ionic Market 130, 131

Ionic Package 135

Ionic Push 131, 132, 133

**Ionic Codepen Demos**

about 285

examples, copying from 285, 286

URL 285

**Ionic community**

about 270

Ionic blog 271

Ionic forum 270

meetup groups 271

Twitter 270

**Ionic Creator**

about 30, 130, 267

HelloWorld app, creating with 281-284

reference 281

URL 267

used, for designing Ionic Project 32, 34

used, for designing prototype 32

**Ionic Deploy**

about 133

URL 134

using 134

**Ionic Factory** 90

**ionic folder, Ionic workflow**

css folder 155

fonts folder 155

js folder 156

scss folder 156

**Ionic forum**

URL 270

**Ionic Framework**

about 14

back menus 61

navigation support 61

URL 271

using, with Brackets 27

using, with different Code Editors 27

using, with Visual Studio 28

**Ionic GitHub**

URL 271

**Ionic header and footer**

about 55

ion-footer-bar directive 56

ion-header-bar directive 56

**Ionic IO**

account, URL 169

URL 170

**Ionic.IO platform**

about 268

URL 268

**Ionic Lab**

about 34, 135

capabilities 136

**Ionic labs technique**

about 167

used, for viewing app 167, 168

**Ionic Market**

about 130, 131

URL 272

**Ionic Modal**

about 215

creating 215

implementing 216, 217

modal.html file, creating 218

wiring up 219, 220

**IonicModal controller**

about 81

initialize(options) 81

**Ionicons 155**

**Ionic Package**

about 135

URL 135

**Ionic Play**

about 27

URL 27

**Ionic Popover**

\$ionicPopover service, adding 211

about 207  
coding 211  
implementing 208, 209  
menu button, adding 209, 210  
popover.html file, creating 212  
wiring up 213, 214

**Ionic Project**  
anatomy 35  
app.js 38  
building 30  
components 37  
folders and files 35  
folder structure 35  
index.html file 37  
ion-content directive 39  
ion-pane directive 39, 40  
root module 38

**Ionic Project, building**  
CDN-hosted library files, using 30, 31  
Ionic CLI, using locally 34  
Ionic Creator, using for prototype design 32, 33

**Ionic Push**  
about 131  
enabling 132, 133

**Ionic SCSS**  
\$ionicConfigProvider 233  
customizing 229, 230, 231  
ionic.app.scss file 225, 226  
overview 222-225  
setting up 226-228  
URL 222

**Ionic serve labs** 167, 168

**ionic serve method** 176

**ionic serve technique**  
about 163, 164  
emulating, with Chrome 164-167

**Ionic services**  
authentication service 88, 89  
versus factories 88

**Ionic setup**  
Ionic CLI, installing 150, 151  
NodeJS, installing 150

**Ionic side menu**  
about 59  
expose-aside-when 60  
<ion-side-menu-content> directive 60

<ion-side-menu> directive 59, 60  
<ion-side-menus> directive 59  
menu-close directive 60  
menu-toggle 60

**Ionic Slack channel**  
about 270  
URL 270

**Ionic social authentication project**  
creating for Facebook, \$firebaseAuth used 457-464

**Ionic starter template**  
about 40  
blank template 40  
maps template 43  
sidemenu template 42, 43  
tabs template 40, 41

**Ionic Tabs**  
about 57  
<ion-tab> directive 58, 59  
<ion-tabs> directive 57, 58

**Ionic tabs application**  
about 235, 236  
creating 236  
extending 251, 252  
overview 239, 240  
post wall feature, implementing 252-254  
running 237  
state for new tab, adding 242, 243  
tab, creating 248, 249  
tabs, adding 242  
tabs.html file 240, 241  
tab-wall.html file, creating 244, 245  
WallController controller, creating 246, 247

**Ionic v2**  
about 136  
migrating to 138  
new features 136

**Ionic View**  
about 168, 301  
benefits 304  
for viewing HelloWorld app 301-304  
URL 169  
used, for testing todo app 169, 170

**Ionic workflow**  
about 154-156  
app, envisioning 158, 159  
app.js file 157, 158

bower.json 156  
config.xml 156  
gulpfile 156  
index.html file 157  
ionic folder 155  
ionic.project 156  
package.json 156  
todo app, building 159

**ion-item**  
about 179  
ion-option-button 180, 181  
ion-option-button element, wiring 188  
ng-repeat, used for rendering list 187

**ion-list application, Bucket-List app**  
implementing 177  
ion-item 179  
ion-list component 178  
ion-option-button 180, 181  
using 178

**ion-pane directive 40**

**ion-spinner 82**

**iOS device**  
todo app, running 171

**iOS Push Notifications**  
URL 132

**issues**  
about 26  
Git command-line tool not installed 26  
npm global modules 26  
permission issue [Mac or Linux] 26

## J

**JS components**  
\$ionicActionSheet 74  
\$ionicBackdrop 75  
\$ionicGesture service 76  
\$ionicLoading 80  
\$ionicModal 81  
\$ionicPopover 82  
about 73  
events 76  
form inputs 75  
gesture events 77  
gestures 76  
ion-spinner 82  
lists 78

**JS files**  
URL 11

**JS SDK**  
URL 95, 96

**L**

**LinkedIn app**  
authentication, configuring in Auth0 465-477  
creating, in Auth0 465

**Linkedin authentication, in Auth0**  
integrating, in Ionic project 478-482

**lists, CSS components**  
about 67  
buttons 68  
dividers 67  
icons 67  
item avatars 68  
thumbnails 68

**lists, JS components**  
about 78  
<ion-list> directive 78

**local JSON database file, London Tourist App (LTA)**  
about 199  
creating 199  
populating 200

**Local Storage**  
to-do app, creating for 370-376

**London Tourist App (LTA)**  
about 191  
creating 191  
data view, designing 201  
developing 199  
Ionic side menu template, using 192  
local JSON database 199  
side menu app, designing 192  
side menu app's code, exploring 194  
side menu app, using 193  
view, designing 201, 202

## M

**maps template 43**

**mBaaS**  
Anypresence 93  
Azure Mobile Services 93

demystifying 93  
Firebase 93  
Kinvey 93  
Parse 93

**Media Capture plugin**  
  URL 341

**meetup groups**  
  URL 271

**menu button**  
  adding, to Ionic Popover 209, 210

**menu.html file, London Tourist App (LTA)**  
  about 195  
  <ion-side-menu-content>element 196, 197  
  <ion-side-menu>element 197, 198  
  <ion-side-menus>element 196

**miscellaneous components**  
  \$ionicConfigProvider 83  
  \$ionicPosition 82

**mobile adoption**  
  considerations 4

**Mobile Apps**  
  about 85  
  developing, with web technologies 8

**mobile development**  
  history 141  
  issues 142, 143

**modules, AngularJS** 9

**multiple views** 51

**multistep form**  
  creating, with validation 316-328

## N

**named view**  
  about 51  
  relative, versus abstract 52

**native mobile applications** 141

**network device plugin**  
  about 122, 123  
  sensors 124

**ngCordova**  
  integrating, to Ionic App 113, 114  
  URL 113, 272, 331

**NodeJS**  
  installing 150  
  URL 150, 271, 276

**Node Package Manager (NPM)** 150

## O

**Options property**  
  allowEdit 115  
  cameraDirection 115  
  correctOrientation 115  
  destinationType 115  
  encodingType 115  
  mediaType 115  
  quality 115  
  saveToPhotoAlbum 115  
  sourceType 115  
  targetHeight 115  
  targetWidth 115

**out-of-the-box filters**  
  reference 429

## P

**Package Control**  
  about 277  
  URL 277

**Parse**  
  Analytics 94  
  API keys, obtaining 95  
  app, creating 94  
  Core 94  
  integrating, REST API used 95  
  integrating, SDK used 95  
  integration 94  
  platform sections 94  
  Push 94  
  setting configuration 95  
  URL 94

**PhoneGap Build**  
  about 510  
  URL 13, 510  
  using, for cross-platform  
    applications 510-512  
  working 511

**photo**  
  capturing, device camera used 332-337

**playground, Ionic**  
  about 269  
  URL 269

**Plugin Manager** 277

**plugins**  
  about 114

camera plugin 114-116  
contacts plugin 121  
Geolocation Cordova plugin 119, 120  
push notifications 117, 118  
**popover.html file**  
  creating 212, 213  
**post wall feature**  
  backend challenge 255  
  implementing 252-254  
**progress bars** 424  
**promise** 390

## Q

**query parameters**  
  \$stateParams service 54  
  about 53  
  multiple parameters 54  
  single parameter 53

## R

**resolve property** 55  
**response object**  
  config property 87  
  data property 87  
  headers property 87  
  status property 87  
  statusText property 87  
**REST API**  
  and \$resource 92  
  used, for Parse integration 95  
  using 96  
**REST API endpoint**  
  URL 133  
**REST principles**  
  URL 96

## S

**SASS**  
  about 146  
  used, for ionic styles 222  
**scroll progress bar directive**  
  creating 424-426  
  working 427, 428  
**SDK**  
  downloading 95

overview 95  
used, for Parse integration 95  
**sensors, network device plugin**  
  device motion 124  
  orientation 125  
**services, AngularJS** 11, 12  
**setCenterLocation() function** 366  
**show method options**  
  buttonClicked 74  
  buttons 74  
  cancel 74  
  cancelOnStateChange 74  
  cancelText 74  
  cssClass 74  
  destructiveButtonClicked 74  
  destructiveText 74  
  titleText 74  
**side menu app's code, London Tourist App (LTA)**  
  exploring 194  
  index.html file 194, 195  
  menu.html file 195  
**sidemenu template**  
  about 42, 43  
  app, creating with 279  
**Single Page Application (SPA)** 8, 145  
**smartphone users**  
  URL 4  
**social networking app**  
  creating, with SQLite 377-392  
**standard templates**  
  about 278  
  blank 278  
  sidemenu 278  
  tabs 278  
**state events**  
  \$stateChangeError 55  
  \$stateChangeStart 54  
  \$stateChangeSuccess 54  
  \$stateNotFound 54  
  about 54  
**state parameters**  
  about 52  
  basic parameters 53  
  query parameters 53  
  Regex parameters 53

**Sublime Text**

about 28, 277  
URL 28, 277

**T****tab interface**

creating, with nested views 308-315

**tabs**

adding, to Ionic tabs application 242  
creating 248, 249  
state, adding 242, 244

**tabs, CSS components 71****tabs template 40, 41****tab-wall.html file**

creating 244, 245

**templates, AngularJS 12****themes**

customizing, for specific platforms 410-412

**todo app**

building 159  
code 160  
creating with ngStorage, for  
  Local Storage 370-377  
Ionic serve labs 167  
ionic serve technique 163, 164  
Ionic view 168  
running, on Android device 170  
running, on devices 170  
running, on iOS device 171  
testing, with Ionic view 169, 170  
UI, creating 159  
user interface, connecting to code 160, 161

**to-do list app**

Bucket-List app, creating 173  
creating 154

**Twitter app**

configuring, with Firebase authentication  
450-453

**types, Hybrid Mobile Application**

Cross compiled 5  
WebView-based 5

**U****UI, Bucket-List app**

designing 174  
input box, implementing 175, 176  
ion-list application, implementing 177  
**updateGroupByUserId() function 391**  
utility, CSS components 73

**V****Velocity.js**

URL 441

**video**

capturing 338-341  
playback, allowing 338-341

**Videogular**

URL 342

**W****WallController controller**

creating 246, 247

**web technologies**

used, for developing Mobile Apps 8

**WebView-based Hybrid Apps 5****Windows Push Notification service**

(WNS) 117





Thank you for buying  
**Ionic : Hybrid Mobile App  
Development**

## About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at [www.packtpub.com](http://www.packtpub.com).

## Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to [author@packtpub.com](mailto:author@packtpub.com). If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles