

A PROJECT REPORT
ON
“HYBRID APPLICATION DEVELOPMENT USING APACHE
CORDOVA & IONIC FRAMEWORK”

Submitted in partial fulfilment of the requirement for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

MANOJ REDDY PANNALA (12K91A05C5)

Under the Esteemed Guidance

Of

Dr. A. SURESH RAO

Professor & Head



Department of Computer Science and Engineering
TKR COLLEGE OF ENGINEERING & TECHNOLOGY

(Affiliated to JNTUH, Hyderabad, approved by AICTE, Accredited by NBA)

Medbowli, Meerpet, Saroornagar, Hyderabad-500097.

April, 2016

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled “**HYBRID APPLICATION DEVELOPMENT USING APACHE CORDOVA & IONIC FRAMEWORK**” carried out by Mr. **MANOJ REDDY PANNALA** bearing Roll No. **12K91A05C5**, in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out by him under my guidance.

The results of investigations enclosed in this report have been verified and found satisfactory. The results embodied in this work have not been submitted to any other University or Institution for the award of any Degree or Diploma.

Guide & HOD

Dr. A. SURESH RAO

EXTERNAL EXAMINER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION BY THE CANDIDATE

I **MANOJ REDDY PANNALA** bearing H.T. No. **12K91A05C5**. I hereby declare that the project report entitled “**HYBRID APPLICATION DEVELOPMENT USING APACHE CORDOVA & IONIC FRAMEWORK**” is done under the guidance of **Mr. Dr. A. SURESH RAO, Professor & Head, Dept. of CSE, TKR College of Engineering and Technology**, is submitted in partial fulfilment of the requirements for the award of the **Degree of Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by me in **TKR College of Engineering and Technology**.

MANOJ REDDY PANNALA (12K91A05C5),
Department of CSE,
TKR College of Engineering and Technology,
Hyderabad.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowned our efforts with success.

I extend my deep sense of gratitude to **Principal, Dr D.V. RAVISHANKAR**, TKR College of Engineering & Technology, for consistent motivation and without whom this could not have been possible.

I am also indebted to the **Head of the Department of CSE, Dr A. SURESH RAO**, for his guidance, constant encouragement and support.

I am indebted to the **Project Coordinators, K. SRINIVAS BABU, Assoc. Professor & P.V. KISHAN RAO, Assoc. Professor**, Computer Science Engineering, TKR College of Engineering & Technology, Meerpet, for their guidance and support.

Finally, I express thanks to one and all that have helped me in successfully completing this seminar. Furthermore, I would like to thank my family and friends for their moral support and encouragement.

MANOJ REDDY PANNALA(12K91A05C5)

INDEX

CONTENTS	PAGE NO
ACKNOWLEDGEMENT	I
ABSTRACT	II
LIST OF TABLES	III
LIST OF FIGURES	IV
LIST OF SCREENSHOTS	V
1. INTRODUCTION	01
2. LITERATURE SURVEY	08
3. SYSTEM ANALYSIS	13
3.1 Existing System	
3.2 Proposed System	
3.3 Software/Hardware Requirements	
3.4 Compilation Chart	
4. SETTING UP ENVIRONMENT	19
4.1 Setting up developer environment	
4.2 Setting up previewing environment	
4.3 Setting up emulators	
4.4 Setting up connected devices	

5. IMPLEMENTATION	28
5.1 Starting a new project	
5.2 Sample code	
5.3 Building the app	
5.4 Build Management	
6. TESTING, PREVIEWING & DEBUGGING	51
6.1 Previewing	
6.2 Debugging	
6.3 Automated Testing	
7. BUILDING AND PUBLISHING APPS	66
7.1 Building Android apps	
7.2 Building iOS apps	
7.3 Screenshots	
8. CONCLUSION	77
9. REFERENCES	78
10. APPENDIX	80

ABSTRACT

Hybrid mobile application is a current thrust area in communication world. In this domain many applications herein referred as apps are developed as user friendly. One such is the TKRCETApp. The significance of the TKRCETApp is, it platform independent with most of the features. This App is built with a combination of web technologies like HTML, CSS, and JavaScript. The key difference is that hybrid apps are hosted inside a native application that utilizes a mobile platform's WebView. (You can think of the WebView as chrome less browser window that's typically configured to run full screen). This enables them to access device capabilities such as the accelerometer, camera, contacts, and more. These are capabilities that are often restricted to access from inside mobile browsers.

TKRCETApp utilizes a consistent set of JavaScript APIs with additional tools of Apache Cordova to access device capabilities through plug-ins, which are built with native code. Also, the current technologies like IONIC frame work and Angular.JS are used to improve the performance.

One build, an application that can run like any other kind of application on the device. The tooling provided by Apache Cordova is largely driven through a command line interface. Also, the security measures are taken into account by using the ionic frame work features. Although, Websites have a restricted set of abilities as opposed to hybrid and native applications which are a constraint features.

1. INTRODUCTION

APACHE CORDOVA:

Apache Cordova (formerly PhoneGap) is a popular mobile application development framework originally created by Nitobi. Adobe Systems purchased Nitobi in 2011, rebranded it as PhoneGap, and later released an open source version of the software called Apache Cordova. Apache Cordova enables software programmers to build applications for mobile devices using CSS3, HTML5, and JavaScript instead of relying on platform-specific APIs like those in Android, iOS, or Windows Phone. It enables wrapping up of CSS, HTML, and JavaScript code depending upon the platform of the device. It extends the features of HTML and JavaScript to work with the device. The resulting applications are hybrid, meaning that they are neither truly native mobile application (because all layout rendering is done via Web views instead of the platform's native UI framework) nor purely Web-based (because they are not just Web apps, but are packaged as apps for distribution and have access to native device APIs). Mixing native and hybrid code snippets has been possible since version 1.9.

The software was previously called just "PhoneGap", then "Apache Callback". As open-source software, Apache Cordova allows wrappers around it, such as Appery.io or Intel XDK.

PhoneGap is Adobe's productized version and ecosystem on top of Cordova. Like PhoneGap, many other tools and frameworks are also built on top of Cordova, including Ionic, Monaca, TACO, the Intel XDK, and the Telerik Platform. These tools use Cordova, and not PhoneGap for their core tools.

Contributors to the Apache Cordova project include Adobe, BlackBerry, Google, IBM, Intel, Microsoft, Mozilla, and others.

DESIGN

The core of Apache Cordova applications uses CSS3 and HTML5 for their rendering and JavaScript for their logic. HTML5 provides access to underlying hardware such as the accelerometer, camera, and GPS. However, browsers' support for HTML5-based device access is not consistent across mobile browsers, particularly older versions of Android. To overcome these limitations, Apache Cordova embeds the HTML5 code inside a native WebView on the device, using a foreign function interface to access the native resources of it.

Apache Cordova can be extended with native plug-ins, allowing developers to add more functionalities that can be called from JavaScript, making it communicate directly between the native layer and the HTML5 page. These plugins allow access to the device's accelerometer, camera, compass, file system, microphone, and more.

However, the use of Web-based technologies leads some Apache Cordova applications to run slower than native applications with similar functionality. Adobe Systems warns that applications may be rejected by Apple for being too slow or not feeling "native" enough (having appearance and functionality consistent with what users have come to expect on the platform). This can be an issue for some Apache Cordova applications.

Within the native application, the application's user interface consists of a single screen that contains nothing but a single web view that consumes the available screen space on the device. When the application launches, it loads the web application's startup page (typically index.html but easily changed by the developer to something else) into the web view, then passes control

to the web view to allow the user to interact with the web application. As the user interacts with the application's content (the web application), links or JavaScript code within the application can load other content from within the resource files packaged with this application or can reach out to the network and pull content down from a web or application server.

The web application running within the container is just like any other web application that would run within a mobile web browser. It can open other HTML pages (either locally or from a web server sitting somewhere on the network). JavaScript embedded within the application's source files implements needed application logic, hiding or unhiding content as needed within a page, playing media files, opening new pages, performing calculations, and retrieving content from or sending content to a server. The application's look and feel is determined by font settings, lines, spacing, coloring, or shading attributes added directly to HTML elements or implemented through Cascading Style Sheets (CSS).

Graphical elements applied to pages can also help provide a theme for the application. Anything a developer can do in a web application hosted on a server can also be done within a Cordova application.

A mobile web browser application does not typically have access to device-side applications, hardware, and native APIs. For example, a web application typically cannot access the Contacts application or interact with the accelerometer, camera, compass, microphone, and other features, nor can it determine the status of the device's network connection. A native mobile application, on the other hand, makes frequent use of those capabilities. For a mobile application to be interesting (interesting to prospective application users, anyway), it will likely need access to those native device capabilities.

Cordova accommodates that need by providing a suite of JavaScript APIs that a developer can leverage to allow a web application running within the Cordova container to access device capabilities outside of the web context. These APIs were implemented in two parts: A JavaScript library that exposes the native capabilities to the web application and the corresponding native code running in the container that implements the native part of the API. The project team would essentially have one JavaScript library but separate native implementations on each supported mobile device platform.

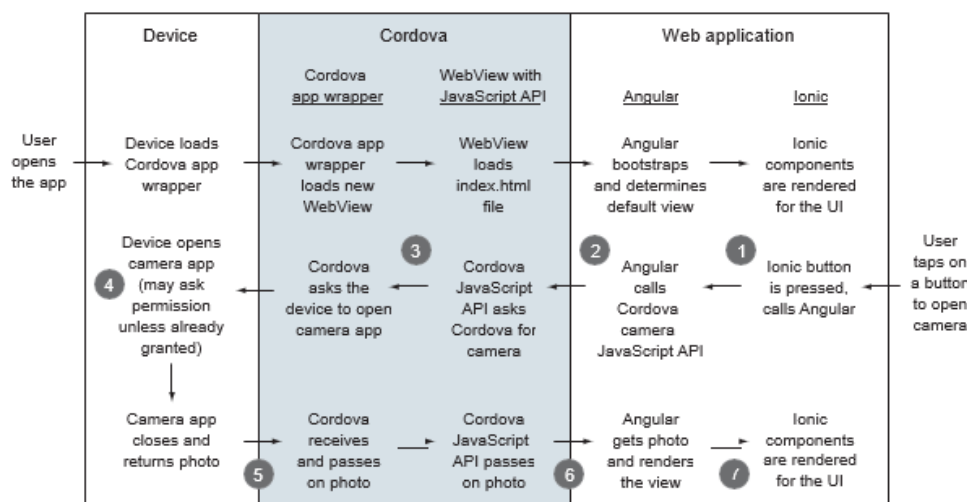


FIGURE 1.1. TYPES OF PLATFORMS

Supported Platforms

Apache Cordova currently supports the following mobile device operating system platforms:

- Android (Google)—<http://developer.android.com/index.html>
- bada (Samsung)—<http://developer.bada.com>
- BlackBerry 10 (BlackBerry)—<https://developer.blackberry.com/>
- iOS (Apple)—<https://developer.apple.com/devcenter/ios/index.action>
- Firefox OS—https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS
- Tizen (originally Samsung, now the Linux Foundation)—<https://developer.tizen.org>
- Windows 8.1---msdn.microsoft.com
- Windows 8 (Microsoft)—<http://msdn.microsoft.com>

Cordova currently provides the following APIs:

- Accelerometer
- Camera
- Capture
- Compass
- Connection
- Contacts
- Device
- Events
- File
- Geolocation
- Globalization
- InAppBrowser
- Media Notification
- Splash screen

IONIC FRAMEWORK

Ionic is a combination of technologies and utilities designed to make building hybrid mobile apps fast, easy, and beautiful. Ionic is built on an ecosystem that includes Angular as the web application framework and Cordova for building and packaging the native app. We'll dig into each in more detail later, but figure shows you an overview of these technologies and how they stack. Let's take a moment to cover the basics of how the technology stack works on a device.

In figure the stack begins with the user opening the app from the device. Imagine this is an iPhone running iOS or a Nexus 10 running Android. Let's break down each of these pieces in more detail:

- **Device**—This loads the app. The device contains the operating system that manages the installation of apps that are downloaded from the platform's store. The operating system also provides a set of APIs for apps to use to access various features, such as the GPS location, contacts list, or camera.
- **Cordova app wrapper**—This is a native app that loads the web application code. Cordova is a platform for building mobile apps that can run using HTML, CSS, and JavaScript inside of a native app, which is known as a *hybrid mobile app*. It's a utility for creating a bridge between the platform and the application. It creates a native mobile app that can be installed.

Ionic stack mental model

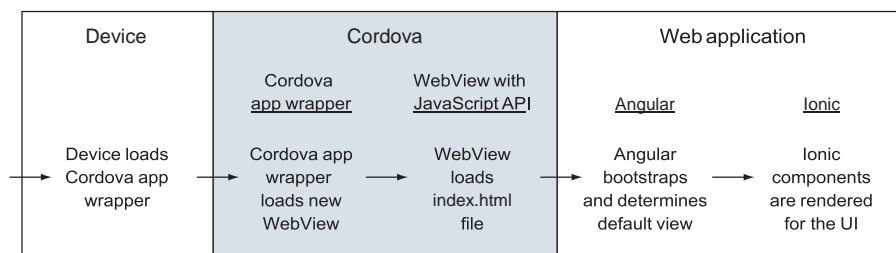


FIGURE 1.2 – IONIC STACK MODEL

and it contains what's called a WebView (essentially an isolated browser window) with a JavaScript API that the web application will run inside.

- **Cordova JavaScript API**—This is the bridge that communicates between the app and the device. The app wrapper has access to both the web application and the native platform through the JavaScript API. This is primarily handled behind the scenes, and Cordova ultimately generates the native app for you.
- **Angular**—This is the web application that controls the app routing and function. The Angular web application runs inside of the WebView. Angular is a very popular framework for building powerful web applications. Angular is primarily used to manage the web application's logic and data.
- **Ionic**—This provides the user interface components rendered in the app. Ionic is built on top of Angular, and is primarily used to design the user interface and experience. This includes the visual elements such as tabs, buttons, and navigation headers. These interface controls are the heart of Ionic, and provide a near-native interface inside of a hybrid app. Ionic also includes a number of additional utilities and features that help manage your app from creation to previewing to deployment.

The combination of these technologies makes Ionic a very feature-rich platform for building your mobile apps. Now that you have a bird's-eye view of Ionic and the technology, let's look a little closer at three main types of mobile experiences and why Ionic's approach is beneficial.

Why Ionic is good for developers

Ionic is able to provide an experience—built into the hybrid app—that looks, feels, and performs like a native app. The long-standing argument that native apps are the only way to get fast and richly featured apps has been proven wrong. People expect their mobile apps to be fast, smooth, and intuitive, and Ionic apps can deliver:

- *Build apps with the web platform*—Using HTML, CSS, and JavaScript, you can make hybrid apps that behave like native mobile apps.
 - *Built with Angular*—For developers familiar with Angular (or even another JavaScript framework like Ember), Ionic is a great choice. Because Ionic is built with Angular, you have access to all of Angular's features and third-party modules. Angular is designed to build major applications, and Ionic extends Angular for the mobile environment.
 - *Uses modern techniques*—Ionic was designed to work with modern CSS3 features like animations. Mobile browsers generally have better support for the latest web platform specifications, which allows you to use those features as well.
- 2 *Engaged community and open source spirit*—The Ionic community is very active on forums, with code contributions, and in sharing tips and tricks about the platform. The open source spirit is alive and well within the project.
 - 3 *Powerful CLI tool*—With the Ionic CLI tool, you can quickly manage development tasks such as previewing the app in a browser, emulating the app, or deploying an app to a connected device. It helps with setting up and starting a project as well.
 - 4 *Ionic services*—Ionic also provides services that make development much easier. The Ionic Creator service allows you to use a drag-and-drop interface to design and export an app. The Ionic View service allows you to deploy an app beta release to customers or test users. In short, Ionic is all about creating not just the basic tools for making hybrid apps, but also the development tools that will help you create them efficiently.
 - 5 *Ionic has a dedicated team*—Open source projects can be difficult to select because you can't be sure if they will be properly developed or supported. Ionic has a dedicated team that has a vested interest in keeping the platform on the leading edge.
 - 6 *Native-like experience*—With Ionic, you can create a look and feel that's like the native apps, making it easier for your customers to use the app.
 - 7 *Performance*—The performance with Ionic is comparable to a native app; the better the app performs, the happier app users will be.
 - 8 *Beautiful, flexible design*—The user interface components have been carefully designed to implement native style guidelines, but also allow for easy customization of any visual aspect of the app.

With Ionic, you can craft feature-rich apps for your customers that take you far less time and effort to

create. This can provide great value for you, your team, and your app users.

Supported mobile devices and platforms

A number of mobile platforms—OS, Android, Windows 8, Firefox OS, Tizen, Black- berry, and more—are in use. With Ionic, you can build for both iOS and Android. Support for Windows 8 and Firefox OS is planned for the future, but isn't currently available.

While it may be possible to develop an app by previewing only on a simulator, devices can act differently in the real world. Let's take a closer look at these two primary platforms and requirements.

Cordova Plugins

As with any developer tool, there are times when the base functionality provided by the solution just isn't enough for your particular needs. Every developer, it seems, wants that one thing that's not already in there. For those cases, the Cordova development team added the ability to extend Cordova applications via plugins. Initially, plugins were hacks inserted in the Cordova container by developers, but over time the Cordova project team solidified a specification for plugins. They even created tools to help developers manage the plugins used in a Cordova application.

With that in place, developers started creating all sorts of plugins. There's a Facebook plugin (<https://github.com/phonegap/phonegap-facebook-plugin>), Urban Airship plugin for push notifications (<http://docs.urbanairship.com/build/phonegap.html>), and one of the most popular plugins, ChildBrowser, actually became a part of the core Cordova API as inAppBrowser. The Cordova development team eventually even migrated all the core Cordova APIs into plugins. I'll show you how to create your own Cordova plugin in Chapter 13, "Creating Cordova Plugins," but for now you can browse a repository of Cordova plugins here: <https://github.com/phonegap/phonegap-plugins>.

2. LITERATURE SURVEY

What Is Adobe Phone Gap?

Adobe PhoneGap is nothing more than an implementation of Apache Cordova with some extra stuff added to it. At its core is the Cordova container and API plugins described in the previous section. As Adobe's primary business is in selling tools and services, the PhoneGap implementation of Cordova more tightly integrates the framework with Adobe's other products.

The primary differences between Cordova and PhoneGap are the command-line tools and the PhoneGap Build service. The PhoneGap command-line tools provide a command-line interface into the PhoneGap Build service.

Throughout the remainder of the book (except in the following "PhoneGap History" section), when I refer to PhoneGap, I'm talking about a specific capability that is available only in the PhoneGap version of Cordova. Both versions are free; PhoneGap simply adds some additional capabilities to Cordova.

Phone Gap History

PhoneGap was started at the 2008 iPhoneDevCamp by Nitobi (www.nitobi.com) as a way to simplify cross-platform mobile development. The project began with a team of developers working through a weekend to create the skeleton of the framework; the core functionality plus the native application container needed to render web application content on the iPhone. After the initial build of the framework, the PhoneGap project team quickly added support for Android with BlackBerry following a short time thereafter.

In 2009, PhoneGap won the People's Choice award at the Web 2.0 Expo LaunchPad competition. Of course, being a project for geeks, the conference attendees voted for the winner by Short Message Service (SMS) from their mobile phones.

Over time, PhoneGap has added support for additional hardware platforms and worked to ensure parity of API features across platforms. During this period, IBM started contributing to the project, as did many other companies.

In late 2011, Nitobi announced that it was donating PhoneGap to the Apache Foundation. Very quickly thereafter, Adobe announced that it was acquiring Nitobi. PhoneGap joined the open source Apache project (www.apache.org) as an incubator project, first as Apache Callback, briefly as Apache DeviceReady, and finally (beginning with version 1.4) as Apache Cordova (the name of the street where the Nitobi offices were located when PhoneGap was created).

The acquisition of Nitobi by Adobe (and Adobe's subsequent announcement that it would discontinue support for Adobe Flash on mobile devices) clearly indicates that Adobe saw PhoneGap as an important part of its product portfolio. The folks at Nitobi who were working on PhoneGap in their spare time as a labor of love are now in a position to work full time on

the project. The result is that Cordova is one of, if not the, most frequently updated Apache projects. The Cordova team delivers new releases monthly, which is pretty amazing considering the complexity of the code involved.

Cordova Going Forward

When you look at the project's description on its Apache project home page (<http://cordova.apache.org/#about>), you'll see that the project team describes itself almost entirely by the APIs Cordova implements.

The Cordova project's efforts around API implementation were initially guided by the World Wide Web Consortium (W3C) Device APIs and Policy (DAP) Working Group (www.w3.org/2009/dap/). This group is working to "create client-side APIs that enable the development of Web Applications and Web Widgets that interact with device services such as Calendar, Contacts, Camera, etc." The plan was for additional APIs to be added as the Cordova project team gets to them and as new standards evolve, but that's not what's happened lately.

From the middle of the Cordova 1.x code stream through the end of the 2.x releases, the project team started working on tightening up the framework. They focused primarily on fixing bugs and cleaning up the project's code. Where there were previously separate JavaScript libraries for each mobile platform, they worked toward consolidating them into a single file (`cordova.js`) and migrating everything from the PhoneGap to the Cordova namespace. For the 3.0 release, the project team focused on stripping the APIs out of the core container and migrating them into separate plugins, then creating some cool new command-line tools to use to manage application projects. The project team should start adding more new APIs to the framework soon after 3.0 is released.

In May 2012, Brian LeRoux (brian.io) from Adobe published "PhoneGap Beliefs, Goals, and Philosophy" (<http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy>) in which he talks about what was (then) driving the project's direction. At the time, as mobile device browsers implemented the DAP APIs in a consistent manner, the plan was for Cordova to obsolete itself. The expectation was that when mobile browsers all support these APIs, there would be no need for the capabilities Cordova provides and the project would just disappear. A good example of this is how modern browsers are starting to add support for the camera, as described in Raymond Camden's blog post "Capturing camera/picture data without PhoneGap" (www.raymondcamden.com/index.cfm/2013/5/20/Capturing-camerapicture-data-without-PhoneGap).

However, one of the things I noticed as I finished PhoneGap Essentials (www.phonegapessentials.com) was that plugins were gaining in prominence in the Cordova space. The APIs provided by Cordova were interesting and helpful to developers, but developers wanted more. Where there were first only a few Cordova plugins available, now there are many, and the core APIs are plugins as well. So, Cordova becomes at its core just a hybrid container, and everything else is done in plugins.

As the browsers implement additional APIs, the core Cordova APIs will become obsolete, but the Cordova container may live on. Cordova could still obsolete itself, but only in a time when the popular mobile browsers provide a standard interface to native APIs. As each platform's OS and API suite are different, I'm not sure how that would work out. It's the cross-platform development capabilities of Cordova that make it most interesting to the market; there's limited chance the market will expose native APIs to the browser in a consistent enough way to keep cross-platform development viable.

It's important to understand there are several ways to build applications for mobile devices, and each has its strengths and weaknesses. There are three basic types: native apps, mobile websites, and hybrid apps. We'll look at each of these in detail to clarify the differences.

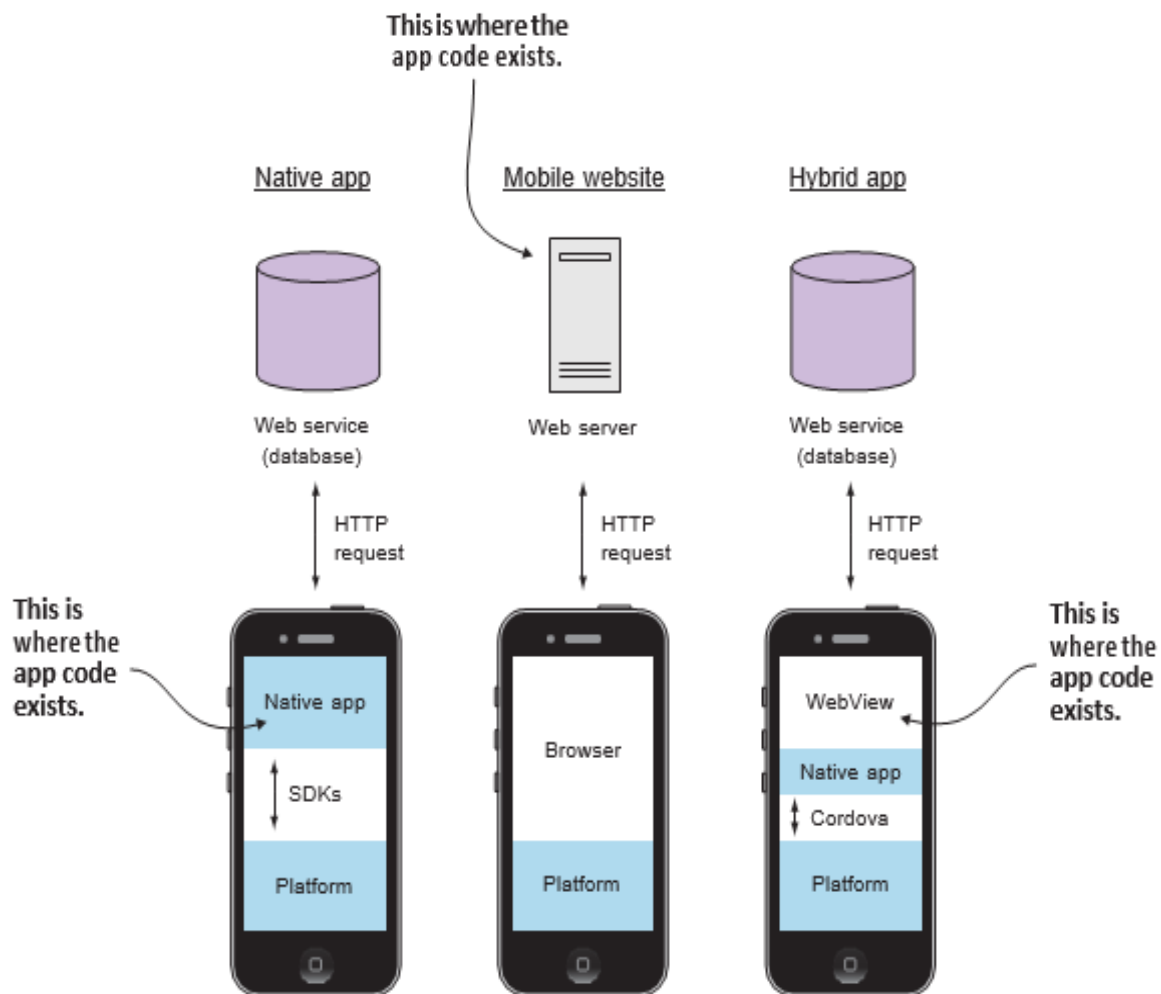


FIGURE 2.1 Native apps, mobile websites, and hybrid app architectures compared side by side

Putting Cordova to Best Use

There are people who try to categorize the types of apps you should or should not build with a hybrid container. I really don't approve of that approach. The hybrid application approach used by Cordova has strengths and weaknesses—and you have to assess your particular mobile application needs against them and decide on a case-by-case basis whether the Cordova approach is the right one for your application.

Web applications are likely going to be slower than native applications, but an inexperienced developer can easily build a native application that performs poorly (without even trying very hard, it seems). You can read about Facebook ditching HTML5 and switching to native for performance reasons (<http://techcrunch.com/2012/12/13/facebook-android-faster>), but then you can read how Sencha was able to build a suitably fast web version of the Facebook application using its HTML5 framework (www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story).

Hybrid applications may be slower than native mobile applications in most cases; it's just the nature of the technology being used. There are reports out there indicating that in more recent versions of iOS, Apple limited the performance of JavaScript running in a web view (instead of in the browser). On the other hand, many games have been created using Cordova, so if Cordova performs well enough for games, it should be okay for many of the applications you need to write.

A lot of commercial applications available today were built using Cordova; you can find a list of many of the applications on the PhoneGap website at www.phonegap.com/apps. The framework is used primarily for consumer applications (games, social media applications, utilities, productivity applications, and more) today, but more and more enterprises are looking at Cordova for their employee-facing applications as well. There are many commercial mobile development tools with Cordova inside and likely more in the works.

So where should you use Cordova? Use it anywhere you feel comfortable using it. Do some proof-of-concepts of your application's most critical and complicated features to make sure you can implement it in HTML and get the performance you need from the framework.

The issues I see are these:

- Can you implement the app you want using HTML? —There are so many JavaScript and CSS frameworks out there to help simplify mobile web development that most things a developer wants to do in an application can be done in HTML.
- Can you get the performance you need from a hybrid application? —That you'll have to prove with some testing.

So where does it fail?

Early on, Cordova would fail when you wanted to build an application that needed access to a native API that wasn't already exposed through the container. Nowadays, with all of the plugins available, you're likely to find one that suits your application's requirements—if not, write your own plugin and donate it to the community. Cordova doesn't (today) have access to the Calendar or email client running on the device, so if your application has requirements for those features, you may be out of luck—but don't forget about plugins.

If your application needs to interface with a particular piece of hardware (either inside the device or outside), Cordova might not be the best choice for you unless there's a plugin for the hardware.

If your application requires heavy-duty offline capabilities such as a large database and offline synchronization, you could run into issues. However, there are several HTML5-based sync engines that should work within a Cordova container, and there are SQLite plugins for Android (<https://github.com/pgsqli/PG-SQLitePlugin-Android>) and iOS (<https://github.com/pgsqli/PG-SQLitePlugin-iOS>).

3. SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

Native mobile apps

To create native apps, developers write code in the default language for the mobile platform, which is Objective C or Swift for iOS and Java for Android. Developers compile the app and install it on a device. Using the platform software development kit (SDK), the app communicates with the platform APIs to access device data or load data from an external server using HTTP requests.

Both iOS and Android provide a set of tools to enable developers to leverage the platform features in a controlled manner through predefined APIs. There are tools, both official and unofficial, that can aid in the development of native apps. It's common for developers to use frameworks in their native app to make development easier.

NATIVE APP ADVANTAGES

Native apps come with a number of benefits over hybrid apps and mobile websites. The benefits revolve around being tightly integrated with the device platform:

Native APIs —Native apps can use the native APIs directly in the app, making the tightest connection to the platform.

Performance —They can experience the highest levels of performance.

Same environment —They're written with native APIs, which is helpful for developers familiar with the languages used.

But there are also a number of disadvantages.

NATIVE APP DISADVANTAGES

The disadvantages of native apps are generally the level of difficulty in developing and maintaining them:

Language requirements —Native apps require developer proficiency in the platform language (for example, Java) and knowledge of how to use platform-specific APIs.

Not cross-platform —They can only be developed for one platform at a time.

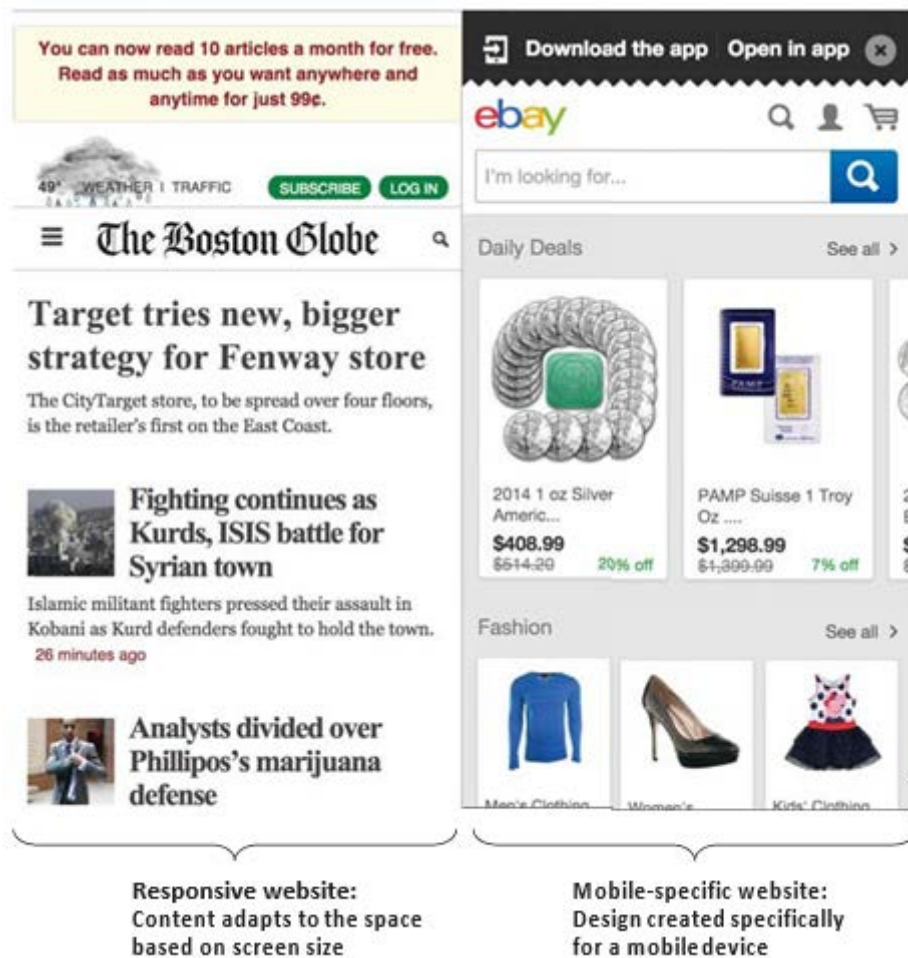
High level of effort —Typically, they require more work and overhead to build, which increases costs.

Native apps may be best suited for developers who have a command of Java and Objective C, or for teams with extensive resources and a need for the benefits of native apps.

Mobile websites (web apps)

Mobile websites, or web apps, work well on a mobile device and are accessed through a mobile browser. Web apps are websites viewed on a mobile device in a mobile browser, designed specifically to fit a mobile device screen size. Figure 1.3 shows a couple of examples.

Some website designers develop a second version specifically for use on a mobile device. Perhaps you've used your mobile device to visit a website and were redirected to a version with limited features, such as visiting eBay and ending up on the <http://m.ebay.com> subdomain. On other websites, such as www.bostonglobe.com, you may



find that the design adjusts to your device's form factor and screen size. This is accomplished with a technique called responsive design. The website content will resize and flow according to the browser window size, and some may even be hidden.

MOBILE WEBSITE ADVANTAGES

Mobile websites enjoy a number of benefits, primarily in the level of effort and compatibility on devices:

Maintainability —Mobile websites are easy to update and maintain without the need to go through an approval process or update installations on devices.

No installation —Because they exist on the internet, they don't require installation on mobile devices.

Cross-platform —Any mobile device has a browser, allowing your application to be accessible from any device.

As with native apps, there are also a number of disadvantages.

MOBILE WEBSITE DISADVANTAGES

Mobile websites run inside of a mobile browser, which is the major cause of limitations and disadvantages:

No native access —Because mobile websites are run in the browser, they have no access to the native APIs or the platform, just the APIs provided by the browser.

Require keyboard to load —Users have to type the address in a browser to find or use a mobile website, which is more difficult than tapping an icon.

Limited user interface —It's difficult to create touch-friendly applications, especially if you have a responsive site that has to work well on desktops.

Mobile browsing decline —The amount of time users browses the web on a mobile device is declining, while app use is increasing.

Mobile websites can be important even if you have a mobile app, depending on your product or service. Research shows users spend much more time using apps compared to the mobile browser, so mobile websites tend to experience lower engagement.

3.2. PROPOSED SYSTEM

HYBRID APPS

A hybrid app is a mobile app that contains an isolated browser instance, often called a WebView, to run a web application inside of a native app. It uses a native app wrapper that can communicate with the native device platform and the WebView. These means web applications can run on a mobile device and have access to the device, such as the camera or GPS features.

Tools that facilitate the communication between the WebView and the native platform make hybrid apps possible. These tools aren't part of the official iOS or Android platforms, but are third-party tools such as Apache Cordova, which is used in this book. When a hybrid app compiles, your web application transforms into a native app.

HYBRID APP ADVANTAGES

Hybrid apps have a few advantages over mobile websites and native apps that make hybrid apps a great platform for building apps:

Cross-platform —You can build your app once and deploy it to multiple platforms with minimal effort.

Same skills as web development —They allow you to build mobile apps using the same skills already used to develop websites and web applications.

Access to device —Because the WebView is wrapped in a native app, your app has access to all of the device features available to a native app.

Ease of development —They're easy and fast to develop, without the need to constantly rebuild to preview. You also have access to the same development tools used for building websites.

Hybrid apps provide a robust base for mobile app development, yet still allow you to use the web platform. You can build the majority of your app as a website, but anytime you need access to a native API, the hybrid app framework provides a bridge to access that API with JavaScript. Your app can detect swipes, pinches, and other gestures just like clicks or keyboard events. But there are a few disadvantages, as you might expect.

HYBRID APP DISADVANTAGES

Hybrid apps have a few disadvantages due to the restrictions that are placed on Web- Views and limitations of native integrations:

WebView limitations—The application can only run as well as the WebView instance, which means performance is tied to the quality of the platform's browser.

Access native features via plugins —Access to the native APIs you need may not be currently available, and may require additional development to make a plugin to support it.

No native user interface controls —Without a tool like Ionic, developers would have to create all of the user interface elements.

With Ionic, you can build hybrid apps so you can leverage the knowledge and skills with which web developers are already familiar.

IONIC STACKFLOW

There are several technologies that can be used when building hybrid apps, but with Ionic there are three primary ones: Ionic, Angular, and Cordova. Figure out- lines how these pieces can work in tandem to facilitate opening the camera from an Ionic app.

1. The user taps on a button (which is an Ionic component).
2. The button calls the Angular controller, which calls Cordova through the JavaScript API.
3. Cordova communicates with the device using native SDKs and requests the camera app.

4. The device opens the camera app (or prompts for permission if necessary), and the user is able to take a picture.
5. When the user confirms the photo, the camera app closes and returns the image data to Cordova.
6. Cordova passes the image data back to the Angular controller.
7. The visual display of the image is updated inside of Ionic components.

This quick outline of how the pieces communicate can demonstrate how an Ionic app is really a stack of technologies that work in concert. Don't worry if some of these terms are unknown to you—we'll cover them throughout this book. The key here is to see how your app is able to leverage the power of the device. Let's look at each one more closely.

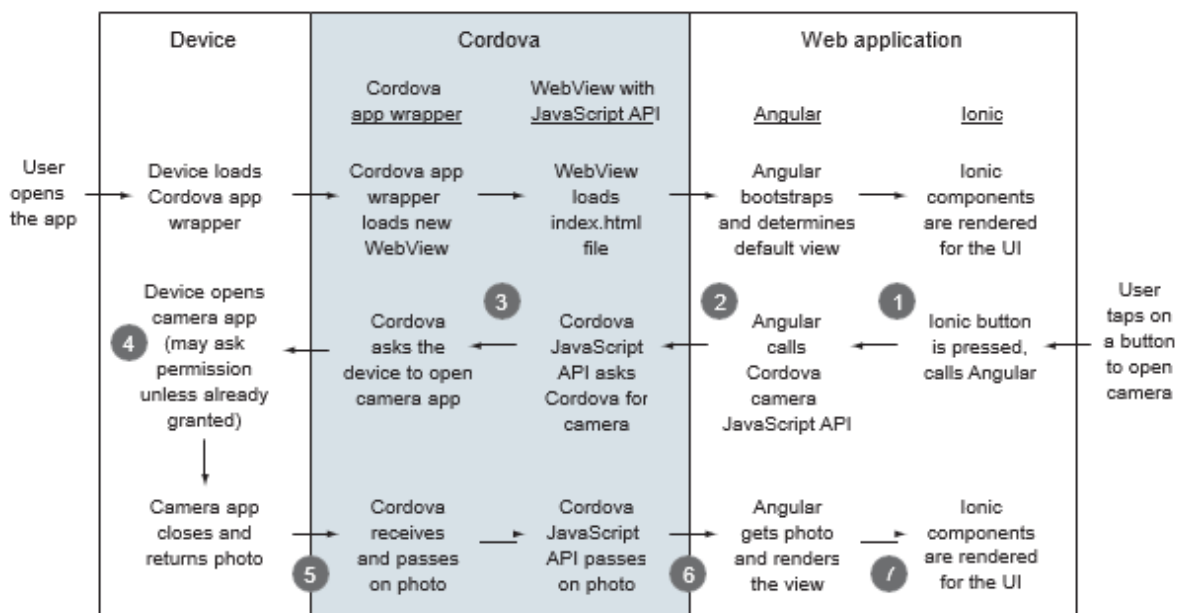


Figure 3.1 How Ionic, Angular, and Cordova work together for a hybrid app

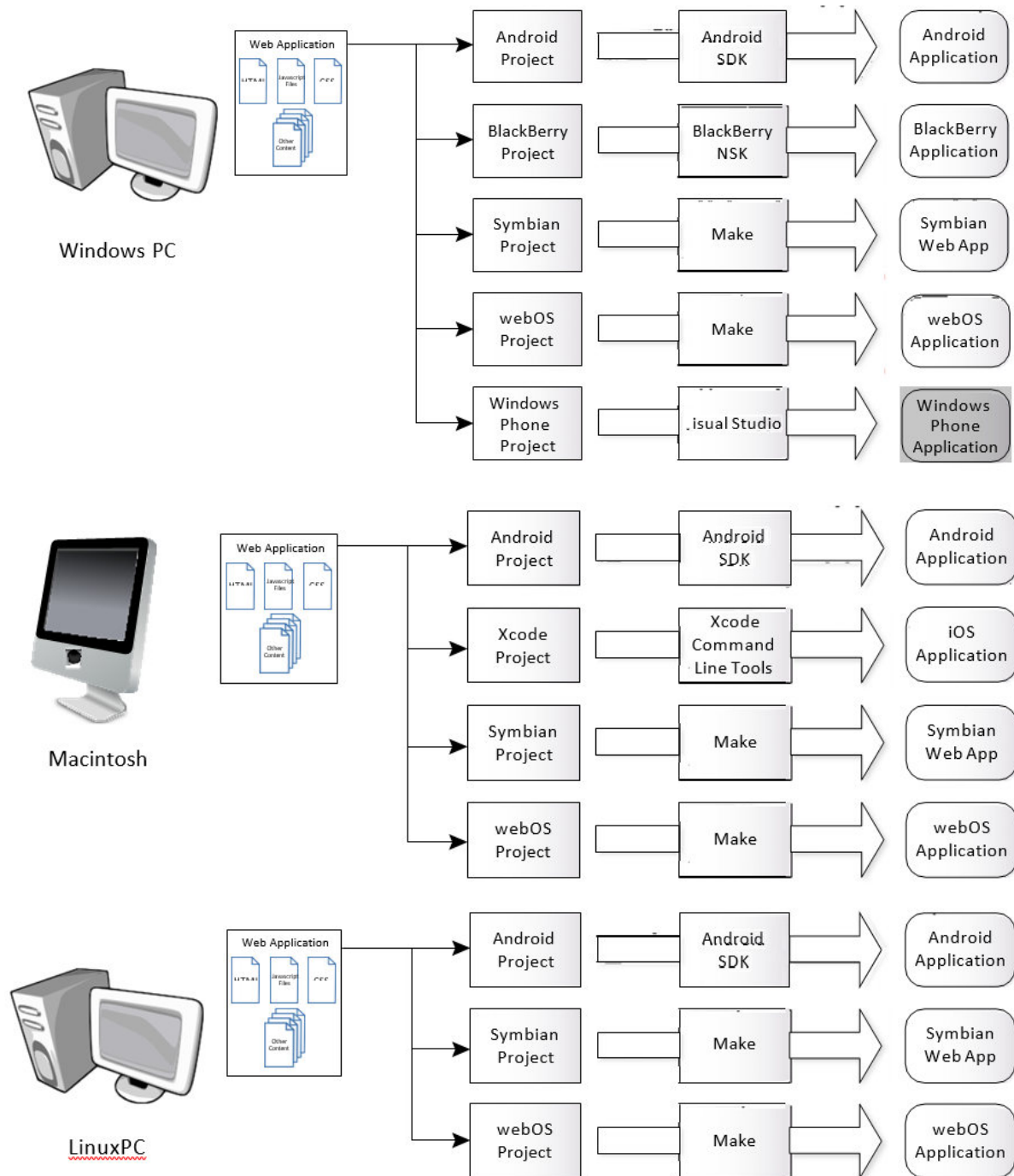
3.3. SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

CPU: AMD FX-6100 SIX-CORE, INTEL CORE-i5.

RAM: 8GB

HARD DISK: 500GB

SOFTWARE REQUIREMENTS:OS: OSX YOSEMITE, WINDOWS 10TOOLS: BRACKETS, SUBLIME TEXT 3IDE: XCODE 6.4, ANDROID STUDIO, WINDOWS VISUAL COMMUNITY**3.4. COMPILATION CHART**

4. SETTING UP THE ENVIRONMENT

4.1. SETTING UP DEVELOPER ENVIRONMENT

To begin building mobile apps with Ionic, you need to ensure you have some required software set up. I'll walk you through how to install and set these up on your computer. In the table, you can see the list of software you need to have installed on your machine to get started.

Optionally, I recommend the use of Git for source code versioning to make it easier to follow along with the source code. This isn't required, but I'll provide you with the

Git commands along the way to follow along more easily. If you're not familiar with Git or if you don't have it installed, you can find more details at <http://git-scm.org>.

If you already have these installed, you can jump to the next section. Otherwise, let's review the installation instructions.

Software	Homepage
Node.js	http://nodejs.org
Ionic CLI	http://ionicframework.com
Cordova	http://cordova.apache.org

INSTALL NODE.JS

Node.js (often referred to as Node) is a platform that runs JavaScript outside of the browser. It allows developers to create applications written in JavaScript that can then execute anywhere. Ionic and Cordova are both written on top of Node, so it's the first installation requirement.

Node can be installed on your machine by going to <http://nodejs.org> and downloading the package for your platform. If you already have Node installed, you should go ahead and install the latest stable version.

You can validate that Node installed correctly by opening a terminal in OS X or the command prompt on Windows and executing the following to check the version of Node:

```
$ node -v v0.12.0
```

If you have any issues with installing Node, you can review the documentation on the Node website. Now we'll use Node's package manager to install Ionic and Cordova.

INSTALL IONIC CLI AND CORDOVA

You can easily install Ionic and Cordova in a single command. This command uses the Node package manager (npm) to install and set up your command-line interface (CLI) tools. Make sure Git is already installed first:

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

This may take a few minutes, depending on the speed of your connection. On a Mac, you may have trouble installing global modules without using sudo. In this case, I'd recommend setting your file permissions correctly for npm so you don't allow Node modules to run as the root user. You can read about how to solve this permission problem at mng.bz/Z97k.

Ionic and Cordova will be installed in such a way that they're available from the command line. Both of these tools execute using Node, but are aliased so you can run them with just the cordova or ionic commands. You can test that they're correctly installed by running the following commands and ensuring they execute without:

```
$ cordova -v 4.2.0
```

```
$ ionic -v 1.3.14
```

Setting up your development environment is important, so make sure each of these are installed and up to date. You should keep Ionic updated, and it will alert you when updates are available. Update Cordova when there are new features you need or bug fixes. Sometimes updating Cordova may require updates to your project, so it should be done only when necessary, and always review the Cordova documentation about possible required changes. To update Ionic or Cordova, you can respectively run the following commands (Ionic will inform you when an update is available):

```
$ npm update -g ionic
```

```
$ npm update -g cordova
```

At this point you have everything you need, so let's start setting up your sample app.

4.2. SETTING UP PREVIEWING ENVIRONMENT

This section will guide you in setting up both emulators and connected devices for previewing your mobile app. Both allow you to preview the app like it's intended to be used, not just in a browser but inside a mobile device. An emulator is a virtual device, which actually runs the mobile platform (Android, for example) in a container and can execute your app like a real physical device. A connected device is any physical device that you connect directly to your computer with a USB cable, and you're able to install your app directly onto it.

To get everything set up, you need to do the following:

Install platform tools needed for building apps.

Download and set up emulators for previewing.

Set up a connected device for previewing.

Set up the project for each supported platform and preview.

The following sections contain a lot of detail about getting started, particularly with Android. Don't get too worried about the apparent complexity, since much of this is a one-time setup. Once you have the tools set up, you'll be able to reuse them for any future projects. You can build prototypes of your app using just the tools we've covered so far in this chapter and come back to this section at a later time when you're ready to start testing on a device.

INSTALLING PLATFORM TOOLS

You need to install additional software to emulate and deploy to connected devices. You only need to set up the software for the platforms you wish to support. Table 2.2 has the required software for Android and iOS development. Ionic version 1.0 only supports Android and iOS fully; other platforms such as Windows Phone or Firefox OS may be supported in future versions.

Platform	Software	Where to find
iOS	Xcode	Search for "Xcode" in App Store on Mac
Android	Android Studio	http://developer.android.com/sdk/index.html

OS X ONLY: INSTALL XCODE FOR IOS

Apple requires XCode for the emulation and distribution of iOS apps. It's only available for Macs, so if you plan to support iOS, then you need to have a Mac.

You can download XCode by opening the App Store and searching for "XCode." It's an official Apple app, and it's quite large (over 3 GB), so be sure to have enough free space.

INSTALL ANDROID STUDIO

Android development can be done on any Windows, Mac, or Linux computer. Android runs on Java, which is cross-platform as well. Android provides two options to choose from: Android Studio or the Android stand-alone SDK Tools. Android Studio is a full IDE with the SDK built in, versus just having the SDK itself. You can download either tool from <http://developer.android.com/sdk/index.html>.

The stand-alone SDK Tools for your platform. Additional installation instructions can be found at <http://mng.bz/flln>.

When you install the stand-alone SDK on Mac or Linux, make sure you add the directory to your path so you can easily execute Android commands. To verify the installation was successful, you can run the following command to see the Android help:

```
android -help
```

Now you're ready to set up emulators.

4.3. SETTING UP EMULATORS

Emulators allow you to run a virtual device on your computer that simulates the actual environment of a mobile device. The virtual device will run the platform inside of the emulator—for example, inside of an Android emulator you can run the actual Android operating system and install your app for development.

You'll want to use an emulator when you're ready to test various types of devices quickly or you need to test your app on a device you don't have access to. It's slower to preview in an emulator compared with the browser, so you'll likely emulate when your app is already functional in the browser.

Emulators require installation and some configuration, and can require some time to download. Let's go over how to set up both Android and iOS emulators.

SETTING UP iOS EMULATOR

Emulators are referred to as simulators in XCode. To begin setting up your iOS simulator, open XCode and then Preferences. In the Downloads tab, you'll see a list of available optional packages, which include documentation and iOS simulators.

I suggest downloading only the most recent simulator at this point. Later you can install the emulators for all versions of iOS that you plan to target for testing. The documentation also isn't necessary because you can find it all online should you need it. Because these simulators and documentation are very large, save yourself the time and disk space and download only what you need.

Once the download is complete, your iOS simulator will be set up and ready to use. You can reset the emulator if you ever need to by having the simulator open and going to the iOS Simulator in the top menu and choosing Reset Content and Settings.

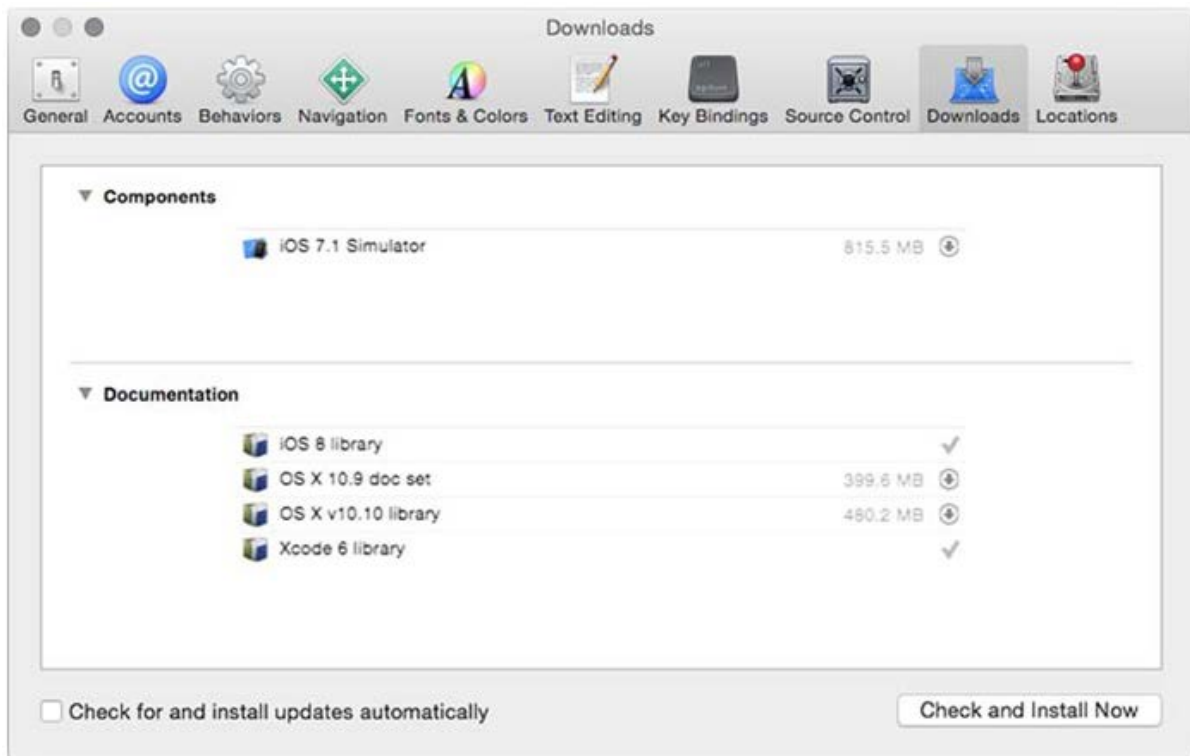


Figure 4.1 In XCode Preferences, use the Downloads tab to download and install iOS simulators.

SETTING UP AN ANDROID EMULATOR

Android emulators are much more free-form than iOS emulators—they allow you to build your own device by declaring the device specifications. Luckily there are some presets that help guide you through this process, but due to the wide variety of Android devices available, setup is a bit more complex than for iOS.

You need to set up the SDK packages, so run android SDK in the command line. The SDK Manager will appear. It allows you to download the platform files for any version of Android, which is a bit more than you need. For the time being, I recommend you download just the most recent release packages and core tools. You need to choose the following items, as shown in figure 4.2:

Tools:

- Android SDK Tools
- Android SDK Platform-tools
- Android SDK Build-tools (choose the most recent version)

Android 4.4.2 (API 19, when paired with Cordova version 4.2):

- SDK Platform
- ARM EABI v7a System Image

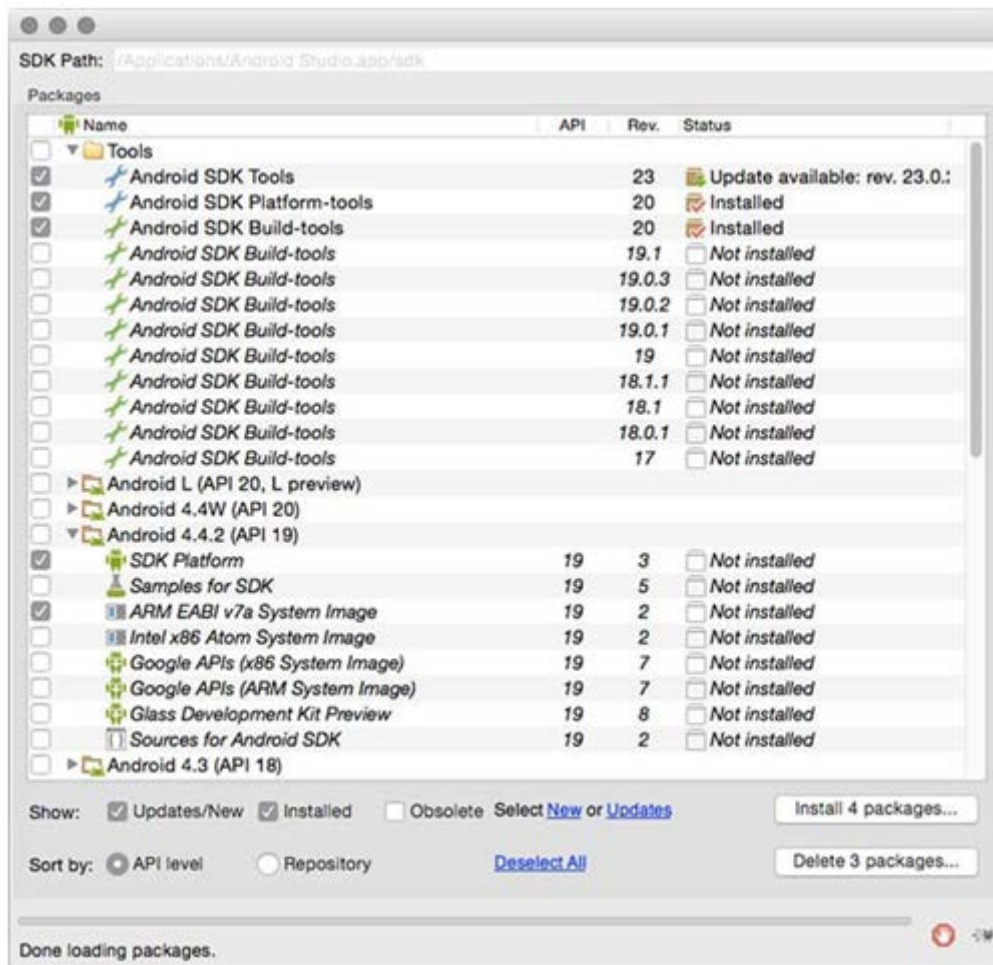


Figure 4.2 Choose the SDK Tools, Platform-tools, and the most recent Build-tools packages, as well as the most recent stable release of Android SDK Platform and the ARM System Image.

Cordova sets a default API level (here, API 19 with Cordova 4.2.0), but that may change over time. You may get a notice later on to install a missing SDK platform for another API level if support changes.

Now you have to define emulator device specifications. This gives you control over the exact device features such as RAM, screen size, and so on. Open the Android Virtual Device (AVD) Manager by executing the following command:

```
android avd
```

Choose the Device Definitions tab so you can set up a device based on a known device configuration, as shown in figure. I recommend using a Nexus 4 or Nexus 5 device, because they're developed by Google and very popular.

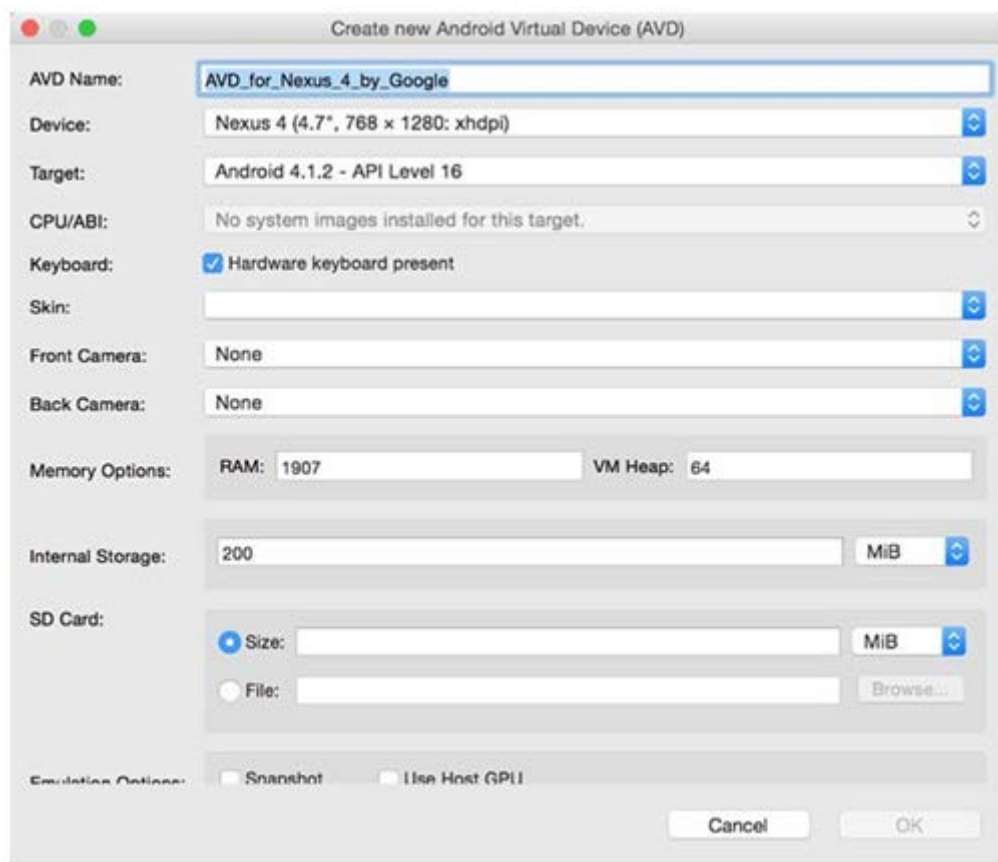
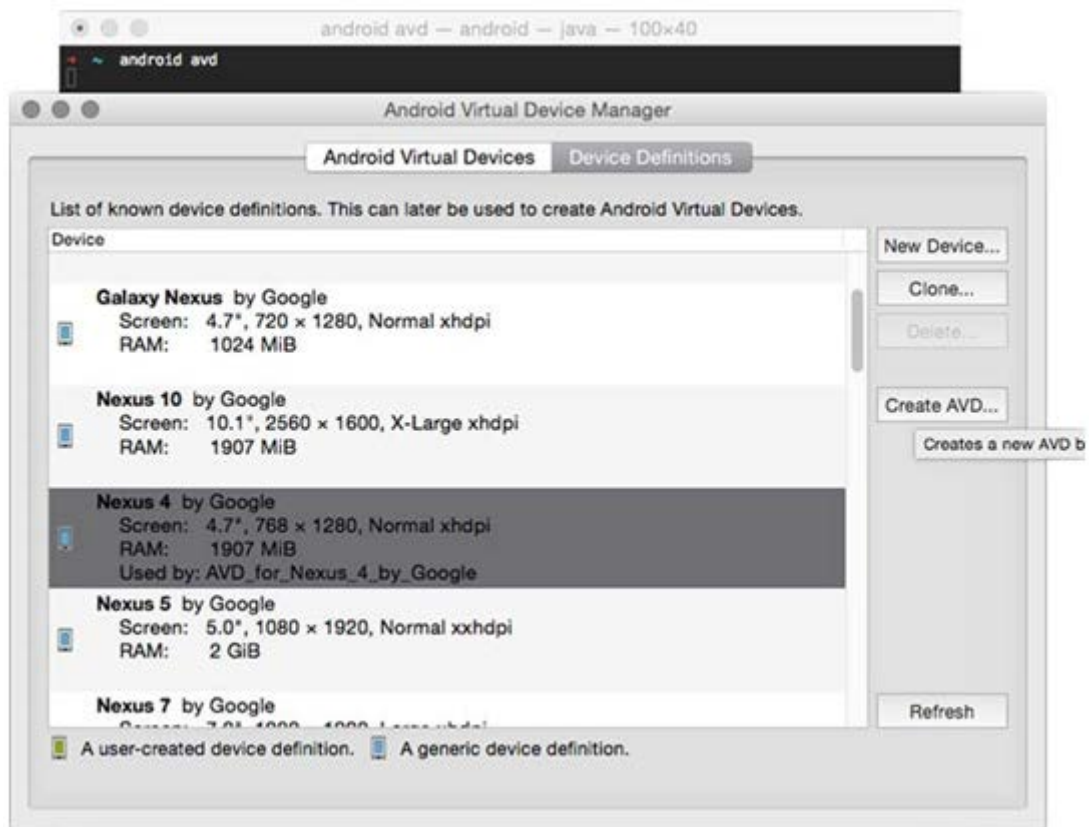


Figure 4.3 Create a new device based on Nexus 4 by Google. This device has the same basic features as an actual phone, though the cameras are not enabled.

Once you've selected the device from the list, click Create AVD and it will open a form with additional details that you can specify about the device. Here you can decide what version of the Android platform to run, the screen size and resolution, and more. Choose the presets like those shown in figure

Once you've finished, click OK and it will save your device. You can create or delete devices as needed, just make sure to always have one set up for emulating. The first time it runs, it may be a bit slow because it will have to do some extra things to set up and boot.

Now that you've got an Android device setup, you can use it in your projects when you want to emulate for Android. When you send an app to your new emulator device, it will boot this device for you.

4.4. SETTING UP CONNECTED DEVICE

If you have an Android or iOS device, you'll want to be able to connect and deploy your apps to it. You can set up any number of connected devices that you have access to, in case you have both new and older devices that you want to be able to test. You'll want to test on devices when possible before you attempt to deploy to a store, and whenever you need to verify the functionality behaves as you expect with a touch environment and that any native plugins work as expected.

SETTING UP AN IOS DEVICE

To connect your iOS device and deploy your app, you have to have an Apple Developer account with iOS. You'll connect your iOS device to your Mac and open XCode. Choose Window > Organizer from the top menu to open the Devices Manager.

Apple requires security profiles to be set up so that your phone is verified to be connected for deploying your apps. You must connect your account in Preferences > Accounts, and it will help you set up certificates and provision profiles. XCode should guide you through the steps for your device because they may vary. For additional assistance, refer to Apple's documentation at https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/. Once the profiles are set up, your device should be available to deploy.

SETTING UP AN ANDROID DEVICE

The first step is to enable developer settings on your Android device. By default, Android devices aren't able to connect to debugging tools unless directly selected by the device owner.

Start by enabling the developer mode on your device, as follows:

1. Open the Settings view and scroll to the last item, About Phone.

2. At the bottom of the About Phone view, there should be a Build Number item—you must tap on it seven times to enable the developer mode. As you get closer to seven taps, the device should notify you how many taps are left.

3. Once this is complete, you can go back to the Settings view and you'll see a new Developer Options item.

Then, to enable USB debugging, you need to do the following steps:

1. Choose the Developer Options item in the Settings view.
2. Scroll down until you see the USB debugging option.
3. Toggle it on—it may prompt you to confirm your choice—and then your device should be ready for debugging when it's connected to your computer.

Now your device is set up to debug, and when it's connected to your computer, the system can detect it for building and deploying to the device.

5. SYSTEM IMPLEMENTATION

IMPLEMENTATION

5.1 STARTING A NEW PROJECT

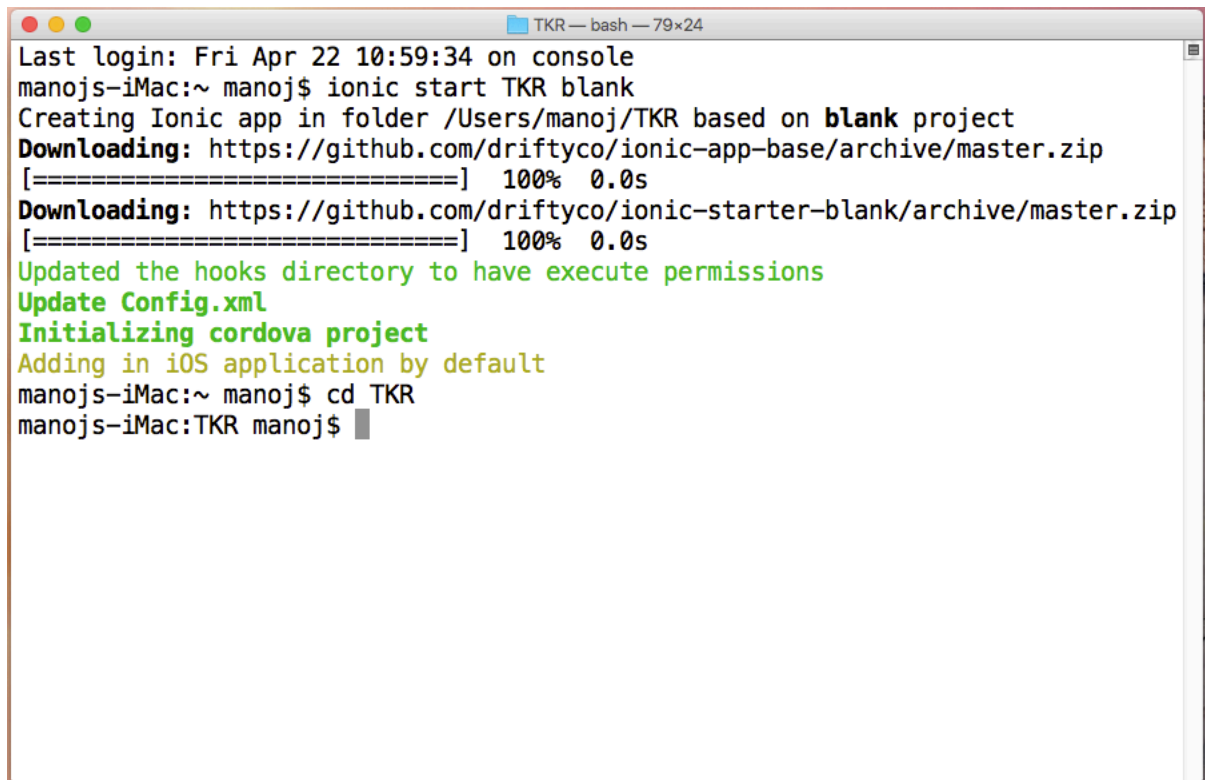
Ionic provides a simple start command that allows you to set up a new project, shown in figure 5.1, in seconds. Ionic provides a set of starter templates that you can use to begin; we'll use the BLANK template here. Run the following commands to create a new project and then to change into the new directory:

```
$ ionic start TKR blank
```

```
$ cd TKR
```

It may ask if you want to create an Ionic account, which you can ignore for now. The account helps you use their services, which we won't be using yet, and you can always create an account later.

Ionic will create a new folder called TKR that will be used to set up the new project using the tabs template. Let's take a moment to understand what each folder is for.

A terminal window titled "TKR — bash — 79x24" showing the execution of the 'ionic start' command. The output includes the creation of the TKR folder, downloading of Ionic app base and starter templates, updating hooks and config, and initializing the Cordova project. The prompt changes to 'manoj\$' in the 'TKR' directory.

```
TKR — bash — 79x24
Last login: Fri Apr 22 10:59:34 on console
manoj$ ionic start TKR blank
Creating Ionic app in folder /Users/manoj/TKR based on blank project
Downloading: https://github.com/driftyco/ionic-app-base/archive/master.zip
[=====] 100% 0.0s
Downloading: https://github.com/driftyco/ionic-starter-blank/archive/master.zip
[=====] 100% 0.0s
Updated the hooks directory to have execute permissions
Update Config.xml
Initializing cordova project
Adding in iOS application by default
manoj$ cd TKR
manoj$
```

FIGURE 5.1 Creating a blank ionic project

Project folder structure

The project folder contains a number of files and directories, which of each have a unique purpose. Here are the files and directories you should see in a new project:

- . bowerrc
- . gitignore
- bower.json
- config.xml
- gulpfile.js
- hooks
- ionic.project
- package.json
- plugins
- scss
- www

This is the generic structure of any Ionic app. The files and directories that are set up and required by Cordova are config.xml, hooks, platforms, plugins, and www. The rest are created by Ionic. Ionic uses both Bower and npm to load some of the dependencies for the project.

The config.xml file is used by Cordova when generating platform files. It contains the information about the author, global preferences, platform-specific preferences, enabled plugins, and more. The default config.xml file generated will use Ionic as the author and HelloWorld as the app name. You can read about all of the options at https://cordova.apache.org/docs/en/edge/config_ref_index.md.html.

The www directory contains all of the web application files that will be run inside of the WebView. It's assumed that there will be an index.html file inside; otherwise, you could structure your files however you like. By default, Ionic sets up a basic AngularJS application that you can build from.

Bower and npm

Both Bower and npm are package management tools that help to download additional files used by a web application. Bower is positioned to help you add additional front-end files to your project, such as jQuery or Bootstrap, and npm is designed to add packages for Node.js projects or Node applications.

With Ionic, the front-end Ionic code is loaded with Bower, and Gulp dependencies are loaded with npm. Gulp is a popular build tool for JavaScript, and we'll discuss Gulp and its role later.

You can find information about Bower at <http://bower.io> and npm at <https://npmjs.org>.

5.1.1 PREVIEWING IN A BROWSER

You can preview your app in the browser, which makes it very easy to debug and develop without having to constantly build the project on a device or emulator. Typically, you'll develop your app using this technique, and then test in the emulator and on a device when the app is more complete. The following command will start a simple server, open the browser, and even auto refresh the browser when you save a file change:

```
$ ionic serve
```

It may prompt you to choose an address, and in most cases you should select local- host. It will open the default browser on your computer on port 8100. You can visit <http://localhost:8100> in any browser, but it's best to preview using a browser used by the platform you're targeting because that's the browser the WebView uses.

Because you're viewing the app in a browser, you have access to the developer tools you'd use for building websites. As you develop, you'll want to have the developer tools open to aid in development and debugging.

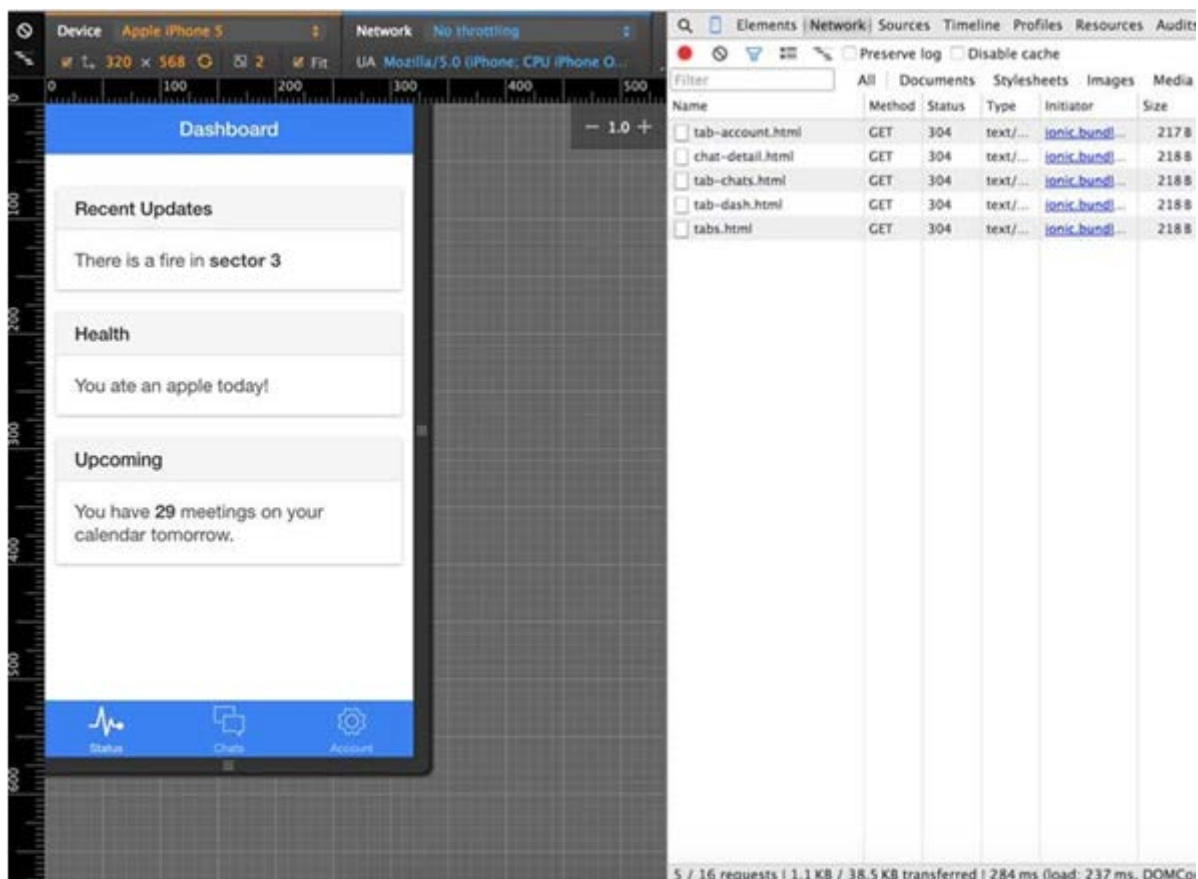
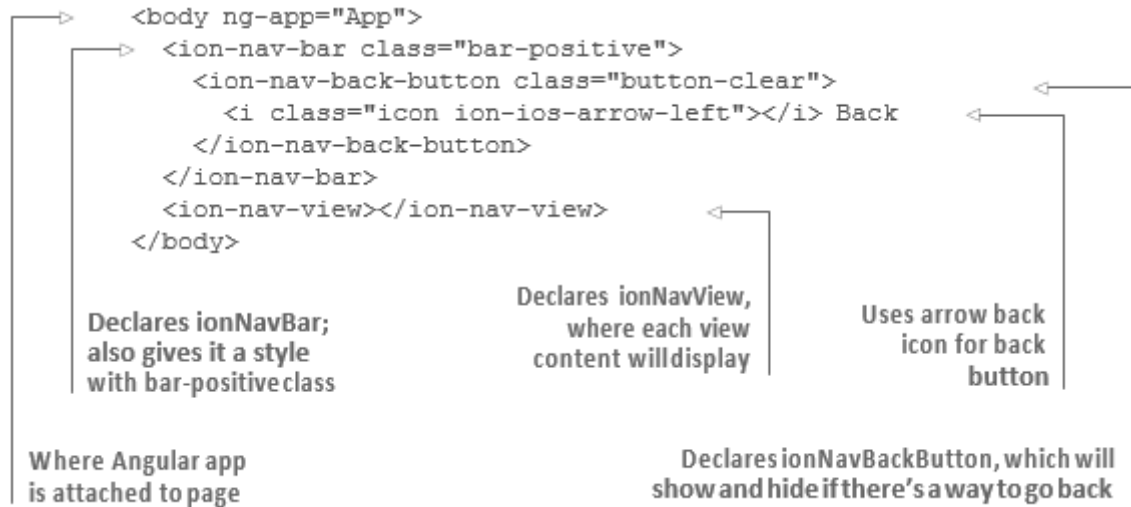


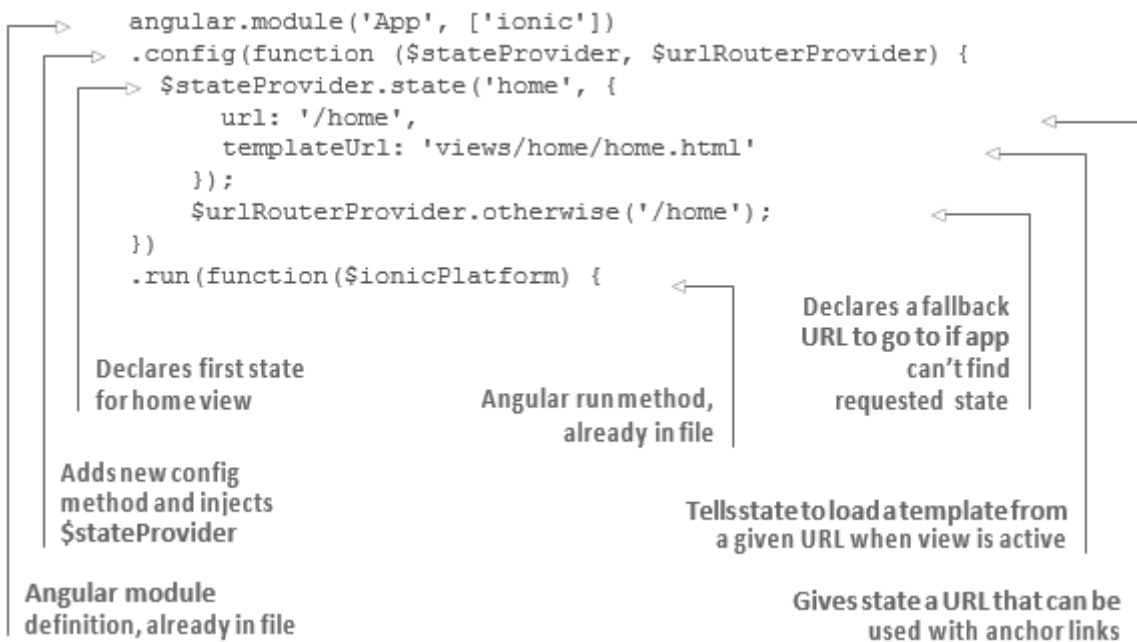
Figure 5.2 Previewing in a Browser

5.1.2 IONIC NAVIGATION AND CORE COMPONENTS

App navigation in markup (www/index.html)



Declaring app states (www/js/app.js)



Adding <ion-content> to home views

```
<ion-view title="TKRCET" id="page1" style="background-color:#607D8B;">
```

```
  <ion-content padding="true" class="has-header">
```

```
    <div style="text-align:center;">
```

```

    </div>

    <div class="spacer" style="width: 278.75px; height:
6px;"></div>

    <ion-list></ion-list>

    <a ui-sref="aboutOurCollege" id="tKRCET-button4"
style="border-radius:-35px -35px -35px -35px;" class="button
button-energized button-block icon-left ion-ios-lightbulb-
outline">About Our College</a>

    <a ui-sref="departments" id="tKRCET-button1"
class="button button-royal button-block icon-left ion-
university">Departments</a>

    <a ui-sref="examBranch" id="tKRCET-button3"
class="button button-calm button-block icon-left ion-
clipboard">Exam Branch</a>

    <a ui-sref="news" id="tKRCET-button12" class="button
button-dark button-block icon-left ion-android-
alert">News</a>

    <a ui-sref="library" id="tKRCET-button15"
class="button button-positive button-block icon-left ion-ios-
book-outline">Library</a>

    <a ui-sref="events" id="tKRCET-button16" class="button
button-assertive button-block icon-left ion-android-star-
half">Events</a>

    <form class="list"></form>

    <a ui-sref="aboutTheApp" id="tKRCET-button17"
class="button button-balanced button-block icon-left ion-ios-
information">About The App</a>

    <button id="tKRCET-button18" class="button button-
light button-block icon-left ion-android-share-alt">Spread a
Word</button>

    <div class="spacer" style="width: 285px; height:
11px;"></div>

    </ion-content>

</ion-view>

```

5.1.3 Using CSS components and adding a simple list of links

Now that you have a content container, you'll want to add a list of links. Ionic comes with a lot of components that are simply CSS classes applied to elements. If you're familiar with front-end interface frameworks like Bootstrap or Foundation, you'll be quite familiar with the method of adding classes to create visual components. Some of the components are mobile-focused designs for several form elements like check-boxes, a range slider, buttons, and more.

Ionic has a list component, which is a pair of classes for a list and each list item. The list component has a number of style configurations; you'll start with the most basic and then add icons.

Let's add a basic list of links to the app, as shown in listing. While you can use a normal unordered list element, I'll actually show you how to use a div to wrap a list of anchor tags. This is important to note because the CSS styling applied is very complete and will allow you to use the class on different elements.

Using a CSS component normally means just following the component guideline and giving elements the proper CSS classes. We'll look at more-complex lists in the next chapter, but for displaying a simple list of items, this suits our needs quite well. The documentation also shows a number of different list item display types, such as having dividers, thumbnails, or icons.

The list has some links to different URLs that you haven't yet defined. You'll add each view individually, and then the app will be able to navigate to that view. With the item class on the anchor tag, it adopts the list item display.

Adding icons to the list items

The last thing you need to do for this view is add some icons. Ionic comes with a set of icons, called Ionicons, that are bundled by default. Icons are used in many places, so you'll see them frequently. You can view the available icons at <http://ionicons.com>. The icons are actually a font icon, which are custom fonts that replace standard characters with icons and use CSS classes to display the icons. If you'd like to use another font icon library (such as Font Awesome), you should be able to include that without conflicts.

The list component has a special display mode that uses icons. Using an extra CSS class and adding an icon element will generate the desired effect of having the icon displayed to the left of the text in the list item. Let's say you'd like the icon to be to the left of the text. You can finish the home view as you see in the following listing by adding some icons and updating the list item with a new class.


```

<ion-view title="Aloha Resort" hide-back-button="true">
  <ion-content>
    <div class="list">
      <a href="#/reservation" class="item item-icon-left">
        <i class="icon ion-document-text"></i> See your reservation
      </a>
      <a href="#/weather" class="item item-icon-left">
        <i class="icon ion-ios-partlysunny"></i> Current weather
      </a>
      <a href="#/restaurants" class="item item-icon-left">
        <i class="icon ion-fork"></i> Nearby restaurants
      </a>
    </div>
  </ion-content>
</ion-view>

```

Adds item-icon-left class to get desired styling

Adds italics element using icon class with icon designation to transform it to icon

This is the most common way to include an icon, but because it's inside of a list, you need to use the special item-icon-left class to get the display you desire. You could also use item-icon-right to have the icons float to the right side.

Icons are often declared like this: `<i class="icon ion-calendar"></i>`. By default, the italics element is an inline element that would modify the text inside. But you have no text inside, just two CSS classes. The first class, icon, gives the element the base CSS styles for an icon, and the second class, ion-calendar, provides the specific icon to display. Together, the inline element becomes an icon. You can see the icon and the entire home view in figure 5.2.

Now the home view is how you want it, so let's move on to the reservation view, and you'll learn how to display information using a controller.

5.1.4 CONTROLLER

A controller does exactly what it sounds like it does: it controls. More specifically it controls the passing of data from the data layer to the view layer via data binding. When you go to a URL in your app, you usually call a controller and a template. The controller will use the template to build your view using data that it will get after it calls a service. This data is stored into the \$scope variable. The \$scope is an object that contains data defined by the controller used to build the view.

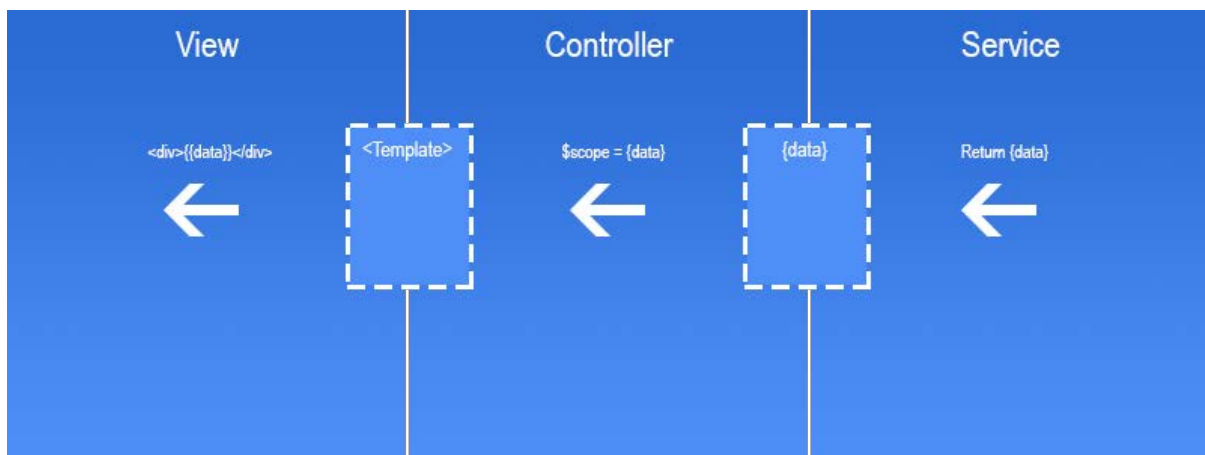


Figure 5.3 Controller Diagram

Building a Basic Controller

So what does a controller look like?

```
.controller('MainCtrl', function($scope) {

})
```

This is about as basic as a controller gets. It's first parameter is the name of the controller followed by the second function parameter that is the body of the controller. Notice the function has a parameter of \$scope. Any parameters we put for the function will be injected by Angular's dependency injection system, assuming they are available by default in Angular or we have imported the module they are in (Read More: [Modules in Ionic/Angular](#)).

So, how do we assign something to the scope? Easy. Just assign to it like you would any variable or object.

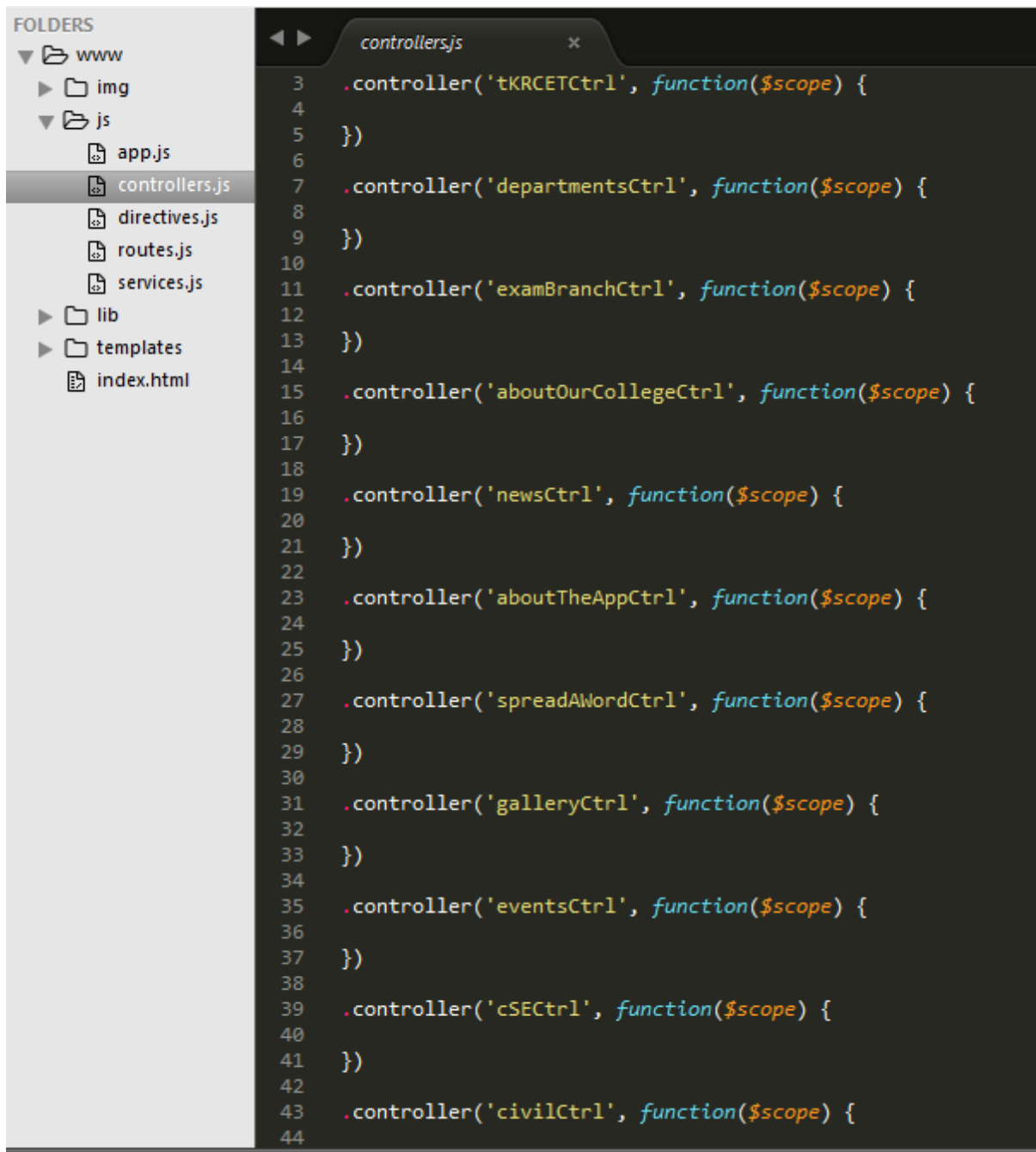
```
.controller('MainCtrl', function($scope) {
    $scope.firstName = "Andrew";
    $scope.lastName = "McGiverty";
})
```

Something else we can do is expose functions to our view by assigning a function to the \$scope.

```
.controller('MainCtrl', function($scope) {
    $scope.add = function(a,b){
        return a + b;
    }

    //or if you prefer...
    function subtract(c,d){
        return c - d;
    }

    $scope.subtract = subtract;
})
```



5.1.5 DEPENDENCIES

The routing functionality added by this step is provided by angular in the ngRoute module, which is distributed separately from the core Angular framework.

We are using Bower to install client-side dependencies. This step updates the bower.json configuration file to include the new dependency:

```
{
```

```

"name": "angular-phonecat",
"description": "A starter project for AngularJS",
"version": "0.0.0",
"homepage": "https://github.com/angular/angular-phonecat",
"license": "MIT",
"private": true,
"dependencies": {
  "angular": "1.4.x",
  "angular-mocks": "1.4.x",
  "jquery": "~2.2.3",
  "bootstrap": "~3.1.1",
  "angular-route": "1.4.x"
}
}

```

The new dependency "angular-route": "1.4.x" tells bower to install a version of the angular-route component that is compatible with version 1.4.x. We must tell bower to download and install this dependency.

If you have bower installed globally, then you can run `bower install` but for this project, we have preconfigured npm to run bower install for us:

`npm install`

Warning: If a new version of Angular has been released since you last ran `npm install`, then you may have a problem with the bower install due to a conflict between the versions of `angular.js` that need to be installed. If you get this, then simply delete your `app/bower_components` folder before running `npm install`.

Note: If you have bower installed globally then you can run `bower install` but for this project we have preconfigured `npm install` to run bower for us.

Multiple Views, Routing and Layout Template

Our app is slowly growing and becoming more complex. Before step 7, the app provided our users with a single view (the list of all phones), and all of the template code was located in the `index.html` file. The next step in building the app is to add a view that will show detailed information about each of the devices in our list.

To add the detailed view, we could expand the `index.html` file to contain template code for both views, but that would get messy very quickly. Instead, we are going to turn the `index.html` template into what we call a "layout template". This is a template that is common for all views in our application. Other "partial templates" are then included into this layout template depending on the current "route" — the view that is currently displayed to the user.

Application routes in Angular are declared via the `$routeProvider`, which is the provider of the `$route` service. This service makes it easy to wire together controllers, view templates, and the current URL location in the browser. Using this feature, we can implement deep linking, which lets us utilize the browser's history (back and forward navigation) and bookmarks.

A Note About DI, Injector and Providers

As you noticed, dependency injection (DI) is at the core of AngularJS, so it's important for you to understand a thing or two about how it works.

When the application bootstraps, Angular creates an injector that will be used to find and inject all of the services that are required by your app. The injector itself doesn't know anything about what `$http` or `$route` services do. In fact, the injector doesn't even know about the existence of these services unless it is configured with proper module definitions.

The injector only carries out the following steps:

- > load the module definition(s) that you specify in your app
- > register all Providers defined in these module definitions

when asked to do so, inject a specified function and any necessary dependencies (services) that it lazily instantiates via their Providers.

Providers are objects that provide (create) instances of services and expose configuration APIs that can be used to control the creation and runtime behavior of a service. In case of the `$route` service, the `$routeProvider` exposes APIs that allow you to define routes for your application.

Note: Providers can only be injected into config functions. Thus you could not inject `$routeProvider` into `PhoneListCtrl`.

Angular modules solve the problem of removing global state from the application and provide a way of configuring the injector. As opposed to AMD or require.js modules, Angular modules don't try to solve the problem of script load ordering or lazy script fetching. These goals are totally independent and both module systems can live side by side and fulfill their goals.

To deepen your understanding of DI on Angular, see [Understanding Dependency Injection](#).

5.2. SAMPLE CODE

Index.html

```
<!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">
    <script src="lib/ionic/js/ionic.bundle.js"></script>

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

    <!-- IF using Sass (run gulp sass first), then uncomment below and remove the CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->

    <style type="text/css">
      .platform-ios .manual-ios-statusbar-padding{
        padding-top:20px;
```

```

    }
</style>

<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/routes.js"></script>
<script src="js/services.js"></script>
<script src="js/directives.js"></script>

<!-- Only required for Tab projects w/ pages in multiple tabs
<script src="lib/ionicui-router/ionicUIRouter.js"></script>
-->

</head>
<body ng-app="app" animation="slide-left-right-ios7">
<div>
  <div>
    <ion-nav-bar class="bar-stable">
      <ion-nav-back-button class="button-icon icon ion-ios-arrow-back">Back</ion-nav-
back-button>
    </ion-nav-bar>
    <ion-nav-view></ion-nav-view>
  </div>
</div>
</body>
</html>

```

Home.html

```

<ion-view title="TKRCET" id="page1" style="background-color:#607D8B;">
  <ion-content padding="true" class="has-header">

```

```

<div style="text-align:center;">
    
</div>

<div class="spacer" style="width: 278.75px; height: 6px;"></div>

<ion-list></ion-list>

<a ui-sref="aboutOurCollege" id="tKRCET-button4" style="border-radius:-35px -35px -35px -35px;" class="button button-energized button-block icon-left ion-ios-lightbulb-outline">About Our College</a>

<a ui-sref="departments" id="tKRCET-button1" class="button button-royal button-block icon-left ion-university">Departments</a>

<a ui-sref="examBranch" id="tKRCET-button3" class="button button-calm button-block icon-left ion-clipboard">Exam Branch</a>

<a ui-sref="news" id="tKRCET-button12" class="button button-dark button-block icon-left ion-android-alert">News</a>

<a ui-sref="library" id="tKRCET-button15" class="button button-positive button-block icon-left ion-ios-book-outline">Library</a>

<a ui-sref="events" id="tKRCET-button16" class="button button-assertive button-block icon-left ion-android-star-half">Events</a>

<form class="list"></form>

<a ui-sref="aboutTheApp" id="tKRCET-button17" class="button button-balanced button-block icon-left ion-ios-information">About The App</a>

<button id="tKRCET-button18" class="button button-light button-block icon-left ion-android-share-alt">Spread a Word</button>

<div class="spacer" style="width: 285px; height: 11px;"></div>

</ion-content>

</ion-view>

```

Controller.js

```

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

.controller('tKRCETCtrl', function($scope) {
})

.controller('departmentsCtrl', function($scope) {
})

```



```
.controller('examBranchCtrl', function($scope) {
})

.controller('aboutOurCollegeCtrl', function($scope) {
})

.controller('newsCtrl', function($scope) {
})

.controller('aboutTheAppCtrl', function($scope) {
})

.controller('spreadAWordCtrl', function($scope) {
})

.controller('galleryCtrl', function($scope) {
})

.controller('eventsCtrl', function($scope) {
})

.controller('cSECtrl', function($scope) {
})

.controller('civilCtrl', function($scope) {
})

.controller('eCECtrl', function($scope) {
})

.controller('eEECtrl', function($scope) {
})

.controller('iTCtrl', function($scope) {
})

.controller('mECHCtrl', function($scope) {
})

.controller('contactUsCtrl', function($scope) {
})

.controller('reachUsCtrl', function($scope) {
})
```

```
.controller('libraryCtrl', function($scope) {})
```

Routes.js

```
$stateProvider
```

```
    .state('tKRCET', {  
      url: '/page1',  
      templateUrl: 'templates/tKRCET.html',  
      controller: 'tKRCETCtrl'  
    })  
  
    .state('departments', {  
      url: '/page2',  
      templateUrl: 'templates/departments.html',  
      controller: 'departmentsCtrl'  
    })  
  
    .state('examBranch', {  
      url: '/page3',  
      templateUrl: 'templates/examBranch.html',  
      controller: 'examBranchCtrl'  
    })  
  
    .state('aboutOurCollege', {  
      url: '/page4',  
      templateUrl: 'templates/aboutOurCollege.html',  
      controller: 'aboutOurCollegeCtrl'  
    })
```

Cse.html

```

<ion-view title="CSE" id="page10" style="background-color:#607D8B;">
  <ion-content padding="true" class="has-header">
    <ion-list>
      <ion-item class="item-thumbnail-left">
        <img>
        <h2>Dr. A. Suresh Rao</h2>
        <p>Professor & Head</p>
      </ion-item>
    </ion-list>
    <div class="list card">
      <div class="item item-divider">Syllabus Copy</div>
      <div class="item item-body">
        <button id="cSE-button11" class="button button-positive button-block button-clear">1st Year</button>
        <button id="cSE-button10" class="button button-positive button-block button-clear">2-1</button>
        <button id="cSE-button13" class="button button-positive button-block button-clear">R13 Academic Regulations</button>
        <button id="cSE-button14" class="button button-positive button-block button-clear">R13 Model Question Paper</button>
      </div>
    </div>
  </ion-content>
</ion-view>

```

5.3. BUILDING THE APP

iOS

For developers building Cordova applications for the iOS platform, Apple provides a suite of tools used to design, package, and deploy iOS applications. Even though the Cordova CLI takes care of most of the process of creating, managing, and testing iOS applications, there will be times when you want to have more control over the process. The CLI can launch a Cordova application in the iOS simulator, but when you encounter problems with an application and you want to know more about what's going on, you need to use the development tools that Apple provides.

ANDROID

Google offers developers a robust suite of tools they can use to develop applications for the Android platform. Even though the Cordova CLI takes care of most of the process of creating, managing, and testing applications, there will be times when you want to have more control over the process. Even though the CLI can launch a Cordova application in the Android platform emulator, when you encounter problems with an application, and you want to know more about what's going on, you'll need to use the development tools that come with the Android Development Tools (ADT).

WINDOWS PHONE

Cordova provides good support for the Windows platform, supporting Windows desktop, Windows Phone 7, and Windows Phone 8. In this documentation, I show you how to use the Microsoft development tools to build and test Cordova applications for Windows Phone 8. The developer tools for Windows Phone are free and pretty easy to use; I found that for the different platforms highlighted in the book, developing for Windows Phone 8 was the easiest.

5.3.1. ADDING DIFFERENT PLATFORMS

iOS

For iPhone devices CLI command used to add the iPhone platform

```
$ ionic platform add ios
```

Android

For Android devices CLI command used to add the Android platform

```
$ ionic platform add android
```

Windows Phone

For Windows devices CLI command used to add the Windows platform

```
$ cordova platform add wp8
```

5.3.2 ADDING DIFFERENT PLUGINS

Plugin Management

The ability to manage a Cordova project's plugin configuration is one of the biggest features of the CLI. Instead of manually copying around plugin files and manually editing configuration files, the CLI does it all for you.

Adding Plugins

To use the CLI to add a plugin to an existing project, open a terminal window, navigate to the Cordova project folder, and issue the following command:

```
cordova plugin add path_to_plugin_files
```

As an example, to add the Camera plugin to a Cordova application, you could use the following command:

```
cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-camera.git
```

This connects to the Apache Git repository and pulls down the latest version of the plugin. You can find a list of the core Cordova plugin locations at http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html#The%20Command-line%20Interface.

In general, the plugin git location shown in the previous example uses the following format:

```
https://git-wip-us.apache.org/repos/asf/cordova-plugin-<plugin-name>.git
```

where <plugin-name> refers to the short name for the plugin.

The CLI can pull plugin code from most any location; if the system can access the location where the plugin files are located, you can install the plugin using the CLI. If you have plugins installed locally, say, for example, if you were working with a third-party plugin or one you created yourself and the files were stored on the local PC, you could use the following command:

```
cordova plugin add c:\dev\plugins\my_plugin_name
```

The command doesn't return anything to the terminal window on success, so unless you see an error reported, the command worked.

Beginning with Cordova 3.1, plugins can be installed directly from the plugin repository just by specifying the plugin name. So, to add the console plugin to your application, you would issue the following command from a terminal window pointing at your Cordova project folder:

`cordova plugin add org.apache.cordova.console`

The CLI checks the repository for a plugin matching the name specified, pulls down the plugin code, and installs it into your project.

It shows a Cordova project folder structure after a plugin has been added using the CLI. Notice that each plugin has a unique namespace built using the Reverse-DNS name convention plus the plugin name. All of the Apache Cordova core plugins are referred to by the plugin name added to the end of `org.apache.cordova.core`. So, for the Camera plugin we just added to the project, you can refer to it via the CLI as `org.apache.cordova.core.camera`. This approach to plugin naming helps reduce potential for name conflicts across similar plugins created by different organizations.

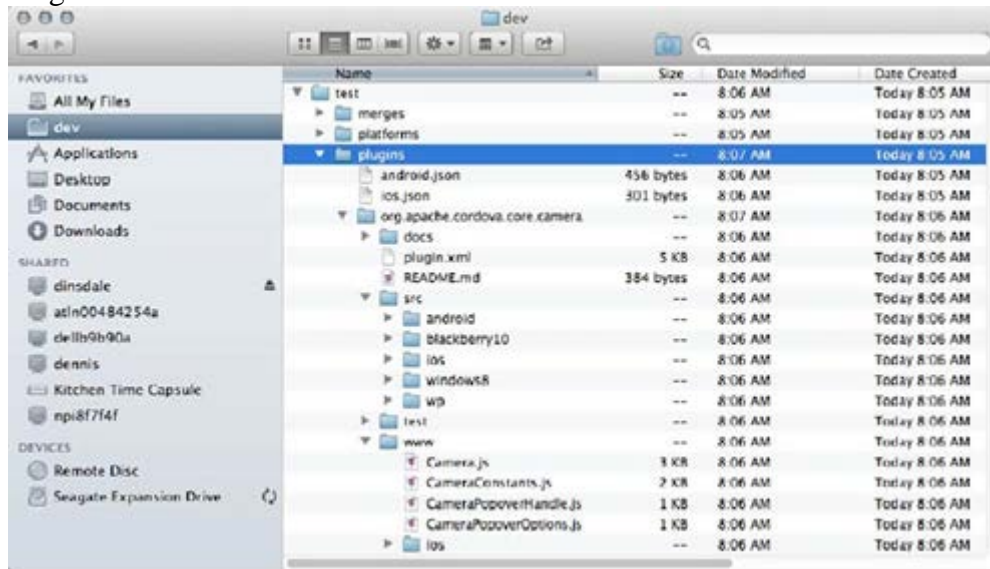


Figure 5.4 Plugins

Listing Plugins

To view a list of the plugins installed in a Cordova project, open a terminal window, navigate to a Cordova project folder, then issue the following command:

```
cordova plugins
```

The CLI will return the list as a JSON array, as shown here:

```
[ 'org.apache.cordova.core.camera', 'org.apache.cordova.core.device-motion',  
'org.apache.cordova.core.file' ]
```

In this case, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

Removing Plugins

To remove a plugin from a Cordova project, open a terminal window, navigate to a Cordova project folder. then issue the following command:

```
cordova plugin remove plugin_name
```

You can also use the shortcut `rm` instead of `remove` to remove plugins:

```
cordova plugin rm plugin_name
```

So, for a project that has the Cordova core File plugin installed, you would remove it from a project by issuing the following command:

```
cordova plugin remove org.apache.cordova.core.file
```

The command doesn't return anything to the terminal window on success, so unless you see an error reported, the command worked.

The CLI will essentially reverse the plugin installation process by removing configuration file entries that point to the plugin plus removing the plugin's folder from the project's plugins folder.

5.4 BUILD MANAGEMENT

The CLI has built-in integration with mobile device platform SDKs, so you can use the CLI to manage the application build process.

Prepare

The CLI prepare command copies a Cordova project's web application content from the www and merges folders into the appropriate platforms folders for the project. This process is described in detail in Chapter 6. You will use this command whenever you make changes to a Cordova web application's content (in the www or merges folder). The prepare command is called automatically before many of the operations described throughout the remainder of this chapter.

To use the prepare command, open a terminal window, navigate to a Cordova project folder, then issue the following command:

```
cordova prepare
```

This copies the web application content into the appropriate folders for each of the mobile device platforms that have been added to the project.

To prepare a specific platform's files, use the following command:

```
cordova prepare platform_name
```

So, to prepare the Android platform folder, use the following command:

```
cordova prepare android
```

The command doesn't return anything to the terminal window on success, so unless you see an error reported, the command worked.

Compile

In this, you can see that there's a cordova folder in each platform's folder structure. Within that folder are platform-specific build scripts used to compile a native application for that platform. The compile command initiates a compilation process by calling the build script for one or more mobile platforms.

To use the compile command, open a terminal window, navigate to a Cordova project folder, then issue the following command:

```
cordova compile
```

To compile a specific platform's native application, use the following command:

`cordova compile platform_name`

So, to compile the Android version of an application, use the following command:

`cordova compile android`

The command doesn't return anything to the terminal window on success, so unless you see an error reported, the command worked.

Build

The build command is similar to compile except that it first calls prepare before calling compile.

To use the build command, open a terminal window, navigate to a Cordova project folder, then issue the following command:

`cordova build`

To build a specific platform's native application, use the following command:

`cordova build platform_name`

So, to build the Android version of an application, use the following command:

`cordova build android`

The command doesn't return anything to the terminal window on success, so unless you see an error reported, the command worked.

Running Cordova Applications

The CLI has built-in integration with mobile device platform simulators, so you can launch Cordova applications directly onto simulators or physical devices. Chapters 7 through 10 provide more detailed information about the simulators, how to configure and launch them, as well as what is required to work with physical devices from the CLI. The following sections provide a high-level overview of the relevant commands.

Emulate

The CLI emulate command automates the process of building an application and deploying it onto a mobile device simulator. The command first prepares the application, executes the build process, then deploys the resulting native application package to the simulator.

To run a Cordova application on the default simulator for each of the platforms configured in the project, issue the following command:

`cordova emulate`

To emulate the application on a single device platform emulator, Android for example, you would issue the following command:

`cordova emulate android`

The CLI is supposed to launch the simulator automatically for you; this works well for iOS but doesn't work for BlackBerry, and I had mixed results trying this on Android.

For the BlackBerry platform, additional steps must be followed to define simulator targets for the emulate command.

Run

The CLI run command automates the process of building an application and deploying it onto a physical device. The command first prepares the application, executes the build process, then deploys the resulting native application package to a connected device.

To run a Cordova application on a physical device for each of the platforms configured in the project, issue the following command:

```
cordova run
```

To run the application on a single device, Android for example, you would issue the following command:

```
cordova run android
```

For the BlackBerry and Windows Phone 8 platforms, additional steps must be followed before you can run an application on a device

Serve

When working with a mobile web application, many developers find that it's better to test the application in a desktop browser before switching to the mobile device. This is especially important when it comes to Cordova applications because an extra packaging process has to happen before the application can be run on a device or simulator.

The CLI includes a serve command, which a developer can use to launch a local web server that hosts a particular platform's web content. It doesn't expose any of the Cordova APIs to the browser, so all you can really test using this option is your web application's UI. You must issue the CLI prepare command before launching the server to make sure your web content is up to date.

To use the serve command, open a terminal window, navigate to a Cordova project folder, then issue the following command:

```
cordova serve platform_name
```

To serve up a Cordova project's Android web application content, you would issue the following command:

```
cordova serve android
```

6. TESTING, PREVIEWING & DEBUGGING

The differences among previewing, debugging, and testing

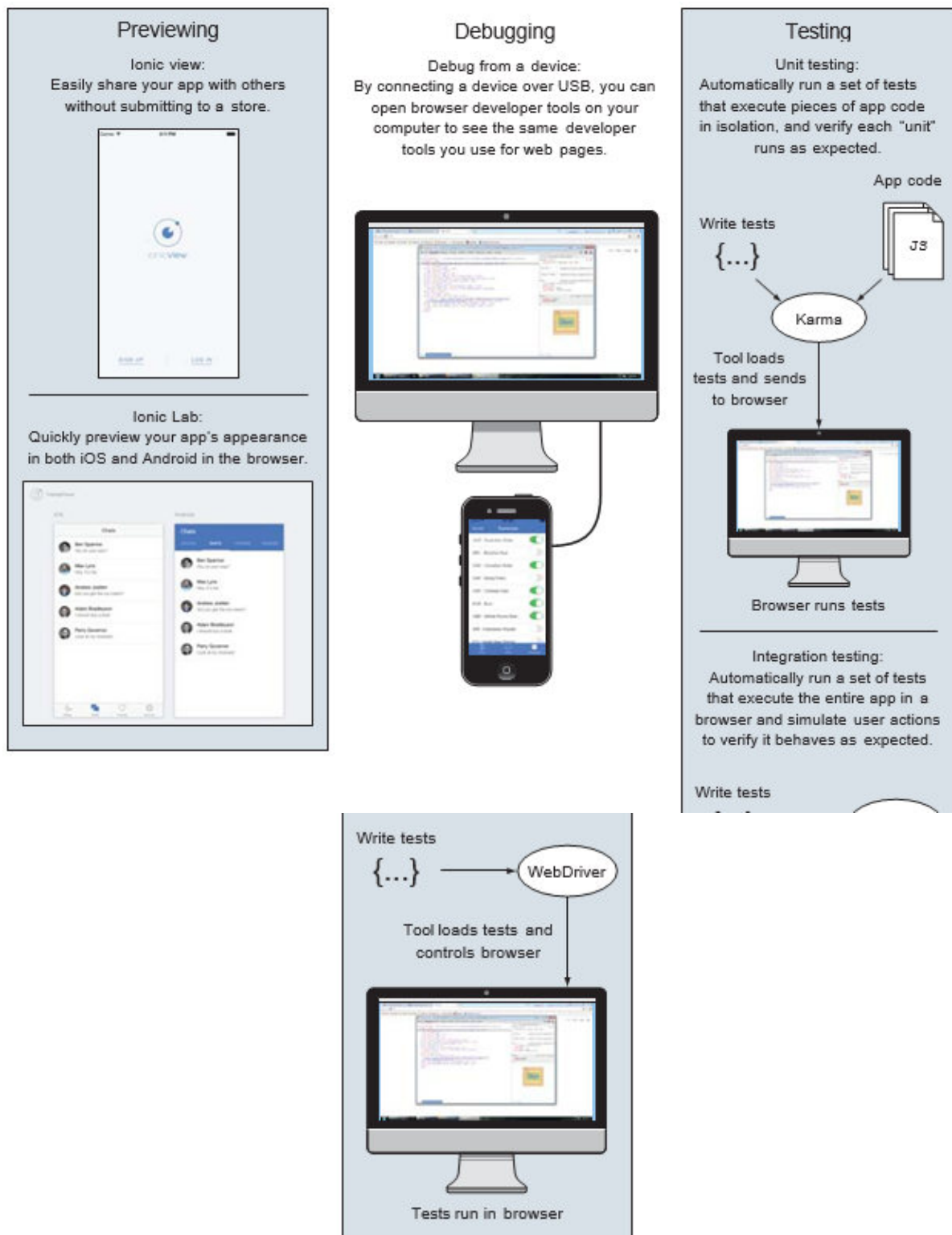


FIGURE 6.1 The differences among previewing, debugging, and testing

Previewing means viewing and interacting with the app on your device or emulator. Previewing is usually the first technique developers use to visually verify how the app looks and that it behaves as expected. Depending on previewing alone can cause a lot of headaches, because the process relies on the developer to manually run and interact with the entire app. As the app becomes larger and the number of platforms increases, manually previewing becomes exponentially more difficult to rely on for quality assurance. You'll look at a few additional ways to preview your app that are built into Ionic.

Debugging is the art of dissecting and discovering the source of a bug. Recall from chapter 1 the technologies and utilities that make up the Ionic stack. Debugging is the act of determining where the error is occurring. It's also possible that some bugs aren't related to your code, but rather something like a corrupt file. Many bugs are resolved today by doing a search online for the error message and finding blog or forum posts that address it. We'll discuss a few techniques and tools that will assist you in tracing your bugs back to their source.

Automated testing is the practice of writing code that can verify the intended behavior of other code. Computers are great at doing repetitive tasks, and testing tools can load your app and execute code to verify it works as expected. Automated tests require that you write a test, which is a way for the test tool to load some code and assert that it does a particular task. Manual testing has a place as well, but automating tests is significantly more practical for production apps.

Why testing is important?

Imagine you have a medium-size app (whatever that means to you) that's for sale on the app stores. You're getting a lot of feedback about a particular problem many people are facing, which you thought you fixed in the past. You need to be able to quickly verify that this bug is fixed before you release a new version of the app. Writing a test is the best way to verify a bug is fixed, because you can run that test repeatedly without having to manually check for the bug in every release.

For web developers who haven't built larger applications, testing may seem like overkill or too much work to implement. Writing a professional, quality app should include testing abilities to maintain quality. But all apps benefit from testing, and you should strive to make this a high priority in your development. Testing has an initial cost to set up, but in the long run it always pays off.

Additional ways to preview apps

There are some useful ways to preview your app that we haven't covered yet, and each is helpful for different types of situations. Ionic is continually creating features for its developers, which is one of the biggest reasons they love it.

You'll look at two additional ways to preview your app besides using ionic serve, ionic emulate, or ionic run. First, with Ionic Lab you can preview your app with both Android and iOS side by side. Second, with Ionic View you can upload an app to the Ionic platform, and others can download and preview your app using the Ionic View app without going through an app store.

6.1 IONIC LAB

When you need to preview the display of your app on iOS and Android at the same time, you can use the Ionic CLI's Lab feature. This technique doesn't require a Mac to preview iOS; however, it's not a real emulator and only provides a visual preview and comparison. It's part of the `ionic serve` command you already know, but when it opens in the browser, you'll see two versions of the app running. In figure 6.2 you can see how one of the views from the chapter 5 example appears with Ionic Lab. This can help you catch bugs related to how the interface appears on different devices.

In figure you can see how on the left, the iOS version, the tabs are displayed on the bottom, and on the right the tabs are displayed at the top. This quickly shows how the appearance of the app differs by platform. To use Ionic Lab, run the `serve` command with the `--lab` flag:

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

This will automatically open a new browser window with the two versions side by side. In chapter 5 we talked about how it's important to design your app to consider the platform's style guides, and this is a great way to quickly preview how your app appears. The same limitations of viewing your app in the browser still apply, so some Cordova features may not work without being in an emulator or on a device.

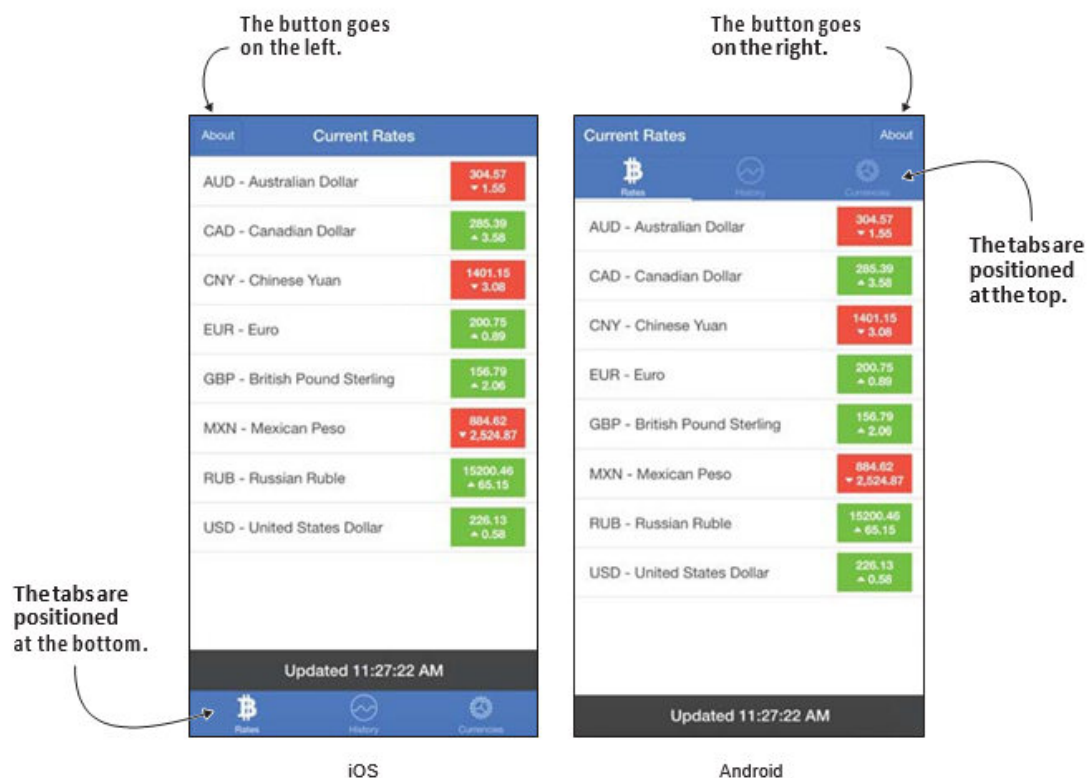


Figure 6.2 Tabs and Side Menus

IONIC VIEW

Ionic has a platform of additional features that Ionic developers can use to make their lives easier. Ionic View (<http://view.ionic.io/>) is a mobile app that anyone can install from the app stores to preview apps made with Ionic. That means you can have clients or beta testers preview your app without actually publishing it to the app stores. For example, you might use it to show your boss the app during your regular progress meetings. IT shows two chapter examples uploaded to my Ionic View app. This doesn't provide you any direct help in developing your app, but it's primarily a way to show the app to others without having to submit it through a store first.

To use Ionic View, you need to have an account with Ionic. You can get a free account at <https://apps.ionic.io/signup>. Then log in from your command line:

```
$ ionic login
```

Fill in your login details there. Once you're authenticated, you'll be able to upload any of your apps to the Ionic platform and share it over Ionic View.

In the command line, navigate to the project directory that you'd like to upload. From there, you can run the command to upload the app, and Ionic will register and upload it to your account:

```
$ ionic upload
```

This command will take any valid Ionic project and send it to the Ionic platform servers. It creates a unique ID for the app and attaches it to your account so you can share it. You can view and manage your uploaded apps at <https://apps.ionic.io/apps>.

After uploading, you can open the Ionic View app on your device, and you should see a list of the apps that you've uploaded. When you tap any app, Ionic View will download and run the app on your device without having to connect the device and deploy it directly.

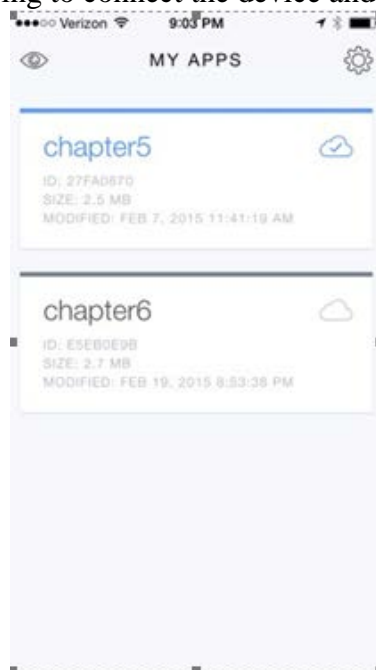


Figure 6.3 Ionic View

IONIC VIEW LIMITATIONS

There are a few limitations to Ionic View. Due to the architecture of the platform, Ionic View can only support a certain set of Cordova plugins. You can view the list of supported plugins in the documentation at <http://docs.ionic.io/docs/view-usage>. Some plugins may not be supported because of security concerns.

Ionic View also doesn't provide debugging information. Production apps are more limited in their abilities to debug for security reasons. Debugging abilities require communication between the app and a computer, which is why you don't want an app you didn't create being able to access your computer directly. You'll need to review the Ionic View documentation to learn about any debugging abilities that might exist should they add features.

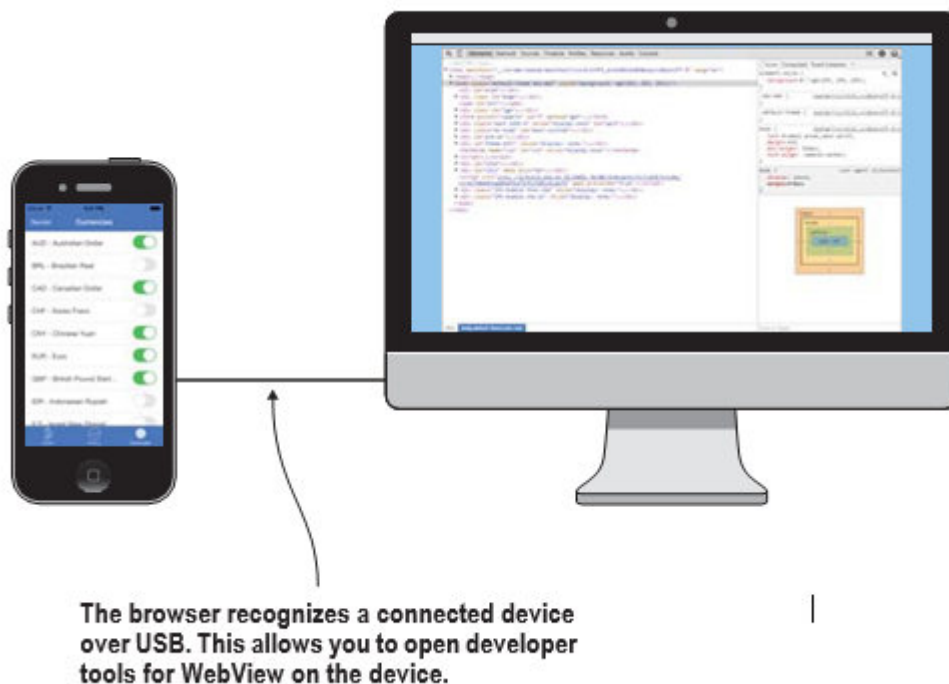


Figure 6.4 Ionic through Connected Devices

6.2 DEBUGGING FROM A DEVICE

So far you've been using the browser on your computer to do development and debugging. But things may happen when you load the app on your device that you need to be able to debug. Because you aren't writing a native app, it would be great to just use the same browser debugging tools that you've been using. The good news is you can!

Both Android and iOS allow you to use the browser developer tools to debug from an emulator or a connected device. Essentially, Chrome or Safari (depending on the platform) allows you to connect to the device and treats the WebView inside of the app as a browser window where you can use developer tools, as you see in figure 9.4.

In chapter 2 we talked about how you can emulate the app or run the app on a device using the Ionic command-line interface (CLI) utility, which has an option to output the console messages into the command line. The following command is for iOS, or substitute ios with android to emulate Android:

```
$ ionic emulate ios -l -c
```

The problem with this is that you only get the JavaScript errors logged into the browser console. This is fine if you need to check for JavaScript errors that you normally see in the browser console, but it provides no ability to inspect the DOM and look at element styles. With the ability to have the complete set of developer tools, you can inspect virtually any aspect of your app. Debugging is only available for apps that you've built and deployed onto a device yourself. Apps aren't designed to be debugged when they're installed from the store.

Debugging from an Android device

Android remote debugging is fairly easy to work with, but requires enabling the debugging options on your device first. Then you can set up the best browser for debugging and get the debugging tools started.

SETTING UP GOOGLE CHROME CANARY BROWSER

It's suggested that you get the Google Chrome Canary browser for Android development. Chrome Canary is the bleeding-edge version of Chrome, and it's intended to allow developers to test new features and changes in Chrome before they go into the primary version most people use. Android development documentation says that for best results when connecting and debugging your app, the browser on your computer should be more advanced than the one installed on the device. Chrome Canary will ensure that's the case because it's like a continuously updated beta version. You can download it from <https://www.google.com/intl/en/chrome/browser/canary.html>.

Once you have a connected device, open Chrome Canary and go to the `chrome://inspect` address. You have to type this into the address bar, and then you'll see the screen in figure 9.5 shown on page 214. If no devices are found, the list will be blank. Because your device is running, you can click the Inspect link to open the Chrome developer tools for the app. You can modify the styles, see the JavaScript console logs, look at network calls such as your API requests, and anything else you can normally do with the developer tools.

Android emulators don't allow you to debug with this technique. But there's another tool called Genymotion that runs like an emulator, but actually appears to the computer as if it's a connected device. You can download and use it for free on personal projects from <https://www.genymotion.com>, and it also requires Virtual Box: <https://www.virtualbox.org>. When you want to deploy your app to Genymotion, you just need to have Genymotion open and then use the `ionic run android` command. If you try to emulate, it will not use Genymotion. That's all you need to do to get access to the debugging tools for Android devices.

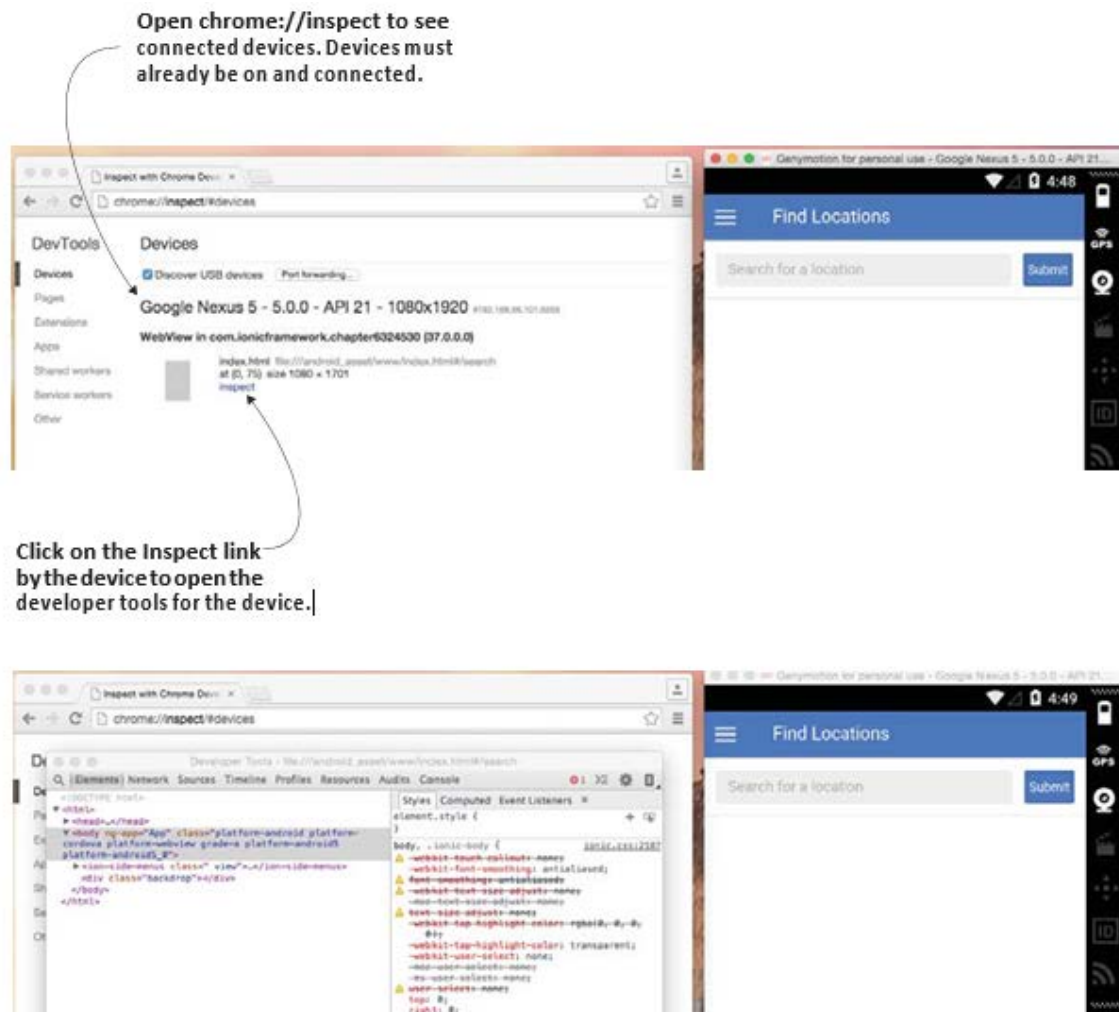
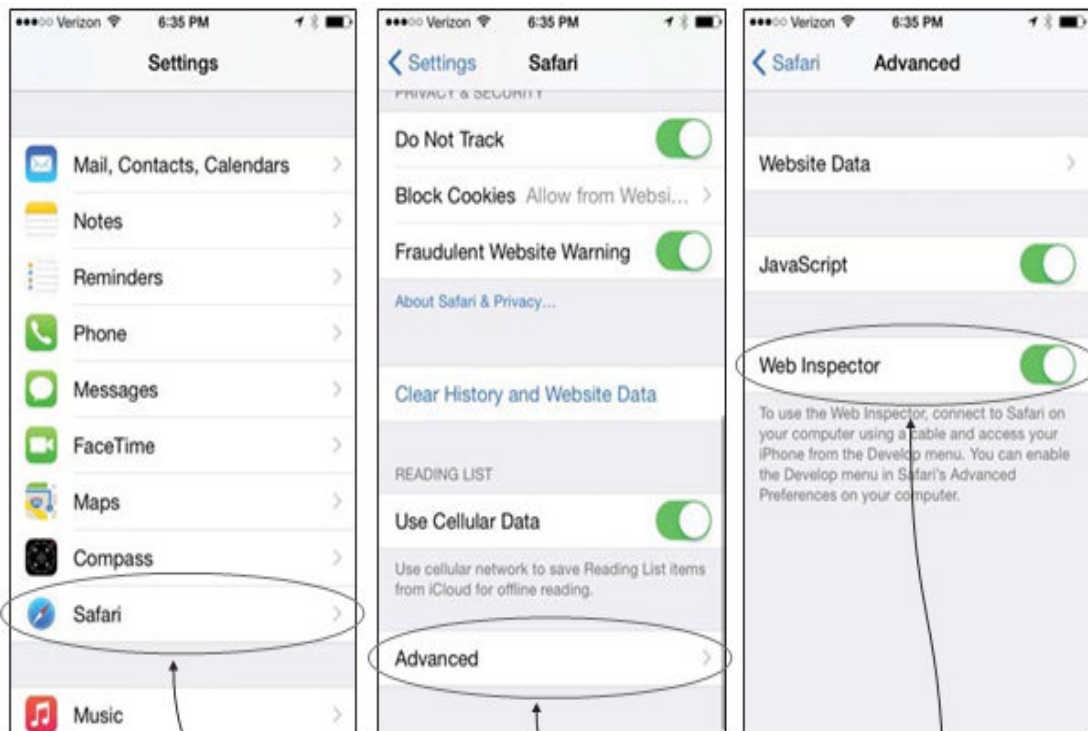


Figure 6.5 Web Inspector

Debugging from an iOS device or emulator

Debugging on iOS is pretty similar to Android, except it uses Safari. Start by enabling debugging through Safari on your device. This should be enabled by default for the iOS emulators, but you can still check the settings to verify. On your device, go to the Settings App. Open the settings for Safari, and then choose the Advanced option at the bottom. Make sure the Web Inspector option is toggled on, which is used by Safari to allow debugging of a web view.

Now open Safari on your computer. If you don't see a Develop menu in the top menu bar, then you'll need to turn on the developer settings for Safari. Open the Safari Preferences panel (Safari > Preferences from the top menu) and choose the Advanced tab. At the bottom of the Advanced tab is a box to check to show the Develop menu. Choose it and close the preferences. Now you can start to debug your app on iOS. First, you'll need to get your app running on an emulator or device. I'll run this chapter example in the emulator, so you can see them in the screenshots side by side with the developer tools.



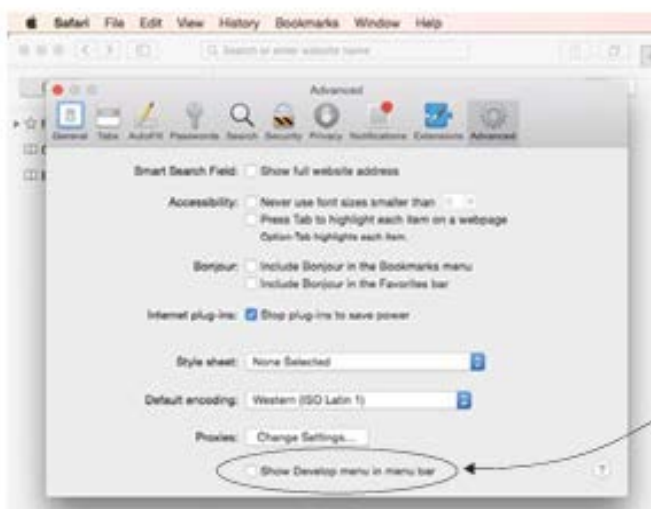
Open Settings and go to Safari.

Select the Advanced option near the bottom.

Toggle the Web Inspector option on to enable debugging.



Open Safari. If you don't have a Develop menu item, then open the Preferences panel.



Select the Advanced tab, and click on the Show Develop menu checkbox at the bottom.

HYBRID APPLICATION DEVELOPMENT USING APACHE CORDOVA & IONIC

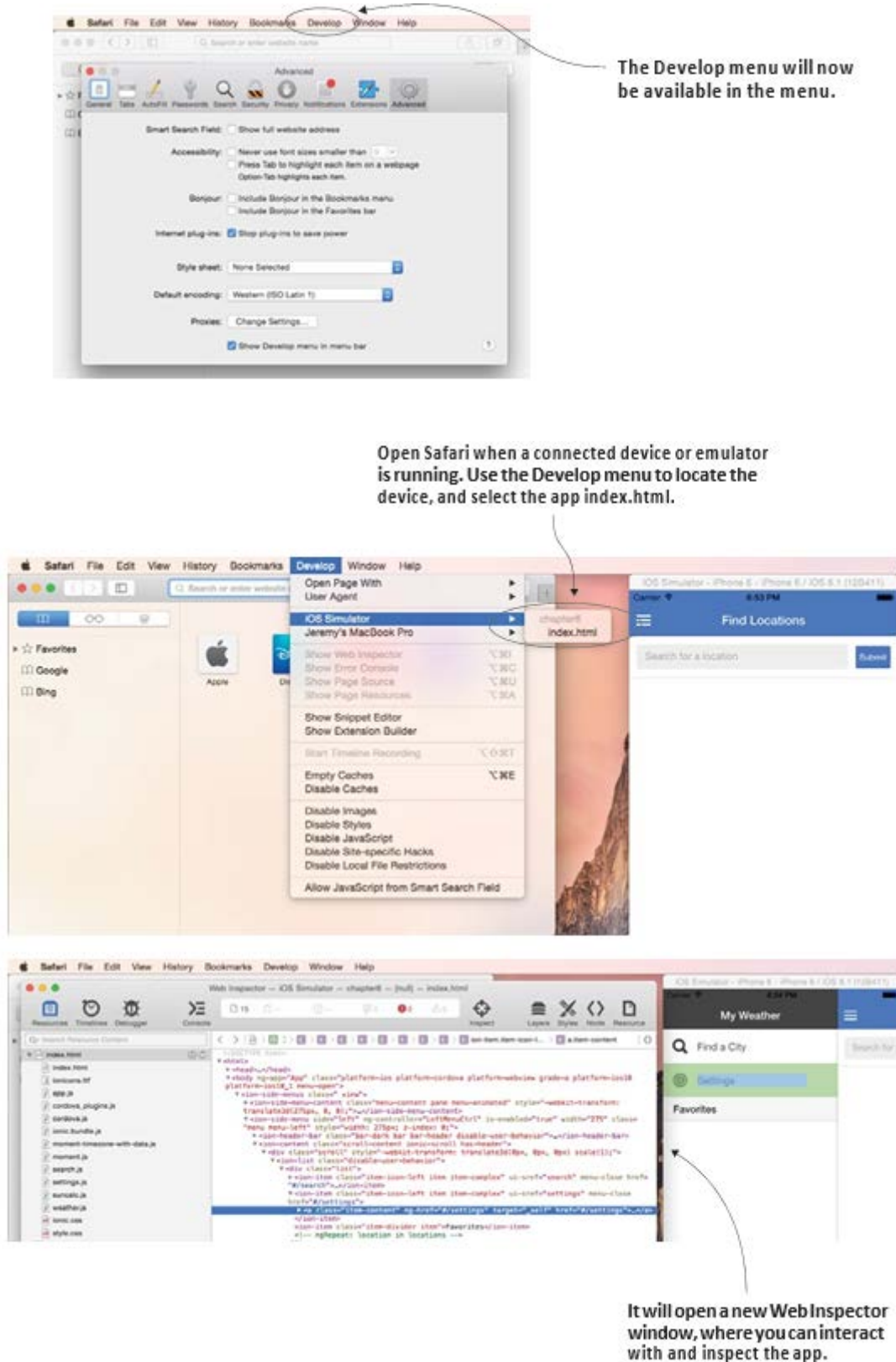


FIGURE 6.6 TESTING

The biggest problem I've run into with debugging on Safari like this is that you have to open the app in the device or emulator before you can open the Web Inspector. That means the Web Inspector can't be opened until the app is already started. If you have any bugs or errors

happening at the moment the app loads, then the Web Inspector will not yet be open to catch the information. You might have to use a hack like an alert to send messages about JavaScript code on load.

That's all there is to setting up debugging for an iOS emulator or device. Now let's dig into setting up automated tests.

6.3 AUTOMATED TESTING

Testing means verifying apps behave as expected. So far, you've been building your app using manual testing by just previewing it and tapping away at the screen. This only works for so long before it becomes cumbersome to manually test every feature, for every platform, for every release.

In a development cycle, you'll use automated testing to help you any time you make a change to your app. You might be fixing a bug you found while debugging or incorporating a new feature that was requested by your client, but you'll want to use automated tests to quickly validate that the app continues to function.

What you want to learn about here is automated testing—code that can programmatically verify if your app is working as expected or not. When done well, these tests can execute in mere seconds, which can take a lot of load off the developer's shoulders. When you work in a team, it also allows others to run tests to verify they didn't break the code of another team member. There are so many good reasons to write tests, so why do some projects not have them?

Simply put, writing tests can be challenging at first. Tests themselves are code, so you have to write code to test code. Developers also might think they can manually test faster than it takes to learn and set up automated testing. But the long-term benefits of automated testing are more considerable: stability in your app, easier development without fear of breaking something, and helping teams avoid conflicts in code.

You'll look at two types of automated tests: unit tests and integration tests (also called end-to-end). The testing tools you'll use work with Angular because your app is based on Angular. Unit tests are best for testing individual parts of your code, such as services and controllers, because a unit test is designed to test each individual function (as its own "unit") to assert it returns the expected value.

Integration tests are designed to test the app behavior as a whole by mimicking user behavior, such as tapping on an item in a list to navigate to a detail view, to verify the interface responds as expected. We'll dig into the nuances of each, but most apps will benefit from both types of tests.

I'll help you get started with the foundations of test writing. By the end of this section, you'll be able to start writing tests, and you'll feel encouraged to dig deeper into the world of testing.

Unit tests with Jasmine and Karma

Unit tests are automated tests for verifying code executes as expected. The intention is to test the smallest parts of the application, such as a scope method, and check that it returns the correct result.

For example, think about your favorite map app. It probably has a method that takes a pair of latitude and longitude values and calculates the distance between the two locations. It would be best to write a set of tests that checks that if you pass the function different types of values (some might even be invalid values), the method returns the expected result. Here are a few fictitious sample tests that might be written to test this conceptual method:

```
var location1 = [91, 21];
var location2 = [82, 32];

expect(mapCalculate(location1, location2)).toEqual(123);
expect(mapCalculate(location1, undefined)).toEqual(0);
```

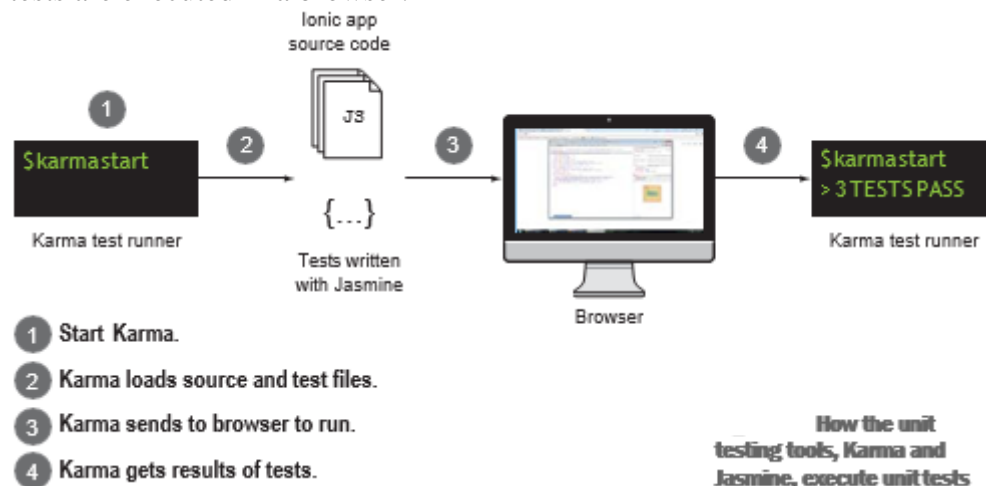
Creates two latitude, longitude values for test to use

Tests that mapCalculate() method gives expected value with valid input

Tests that mapCalculate() method handles invalid input as expected

Unit tests are a great way of ensuring the smallest parts of your app work as intended. If you have confidence that the unit tests are running and that your methods are all working as expected, then it becomes easier to make changes in other parts of the application without fear of breaking existing features. I've learned the hard way that without tests it can become very difficult to maintain an app over time.

You'll use the Jasmine (<http://jasmine.github.io/>) testing framework to write unit tests, and then use Karma (<http://karma-runner.github.io/>) as the tool that will run them. Jasmine is a popular option for developers who are new to testing, and it's also the primary testing framework used by the Ionic and Angular projects. As shown in figure, Karma connects a testing framework (in this case, Jasmine), loads all of the tests and application code into a browser, and then runs the tests in a browser (in this case, Chrome). Because your JavaScript runs in the browser, the Jasmine tests are executed in a browser.



SET UP KARMA AND JASMINE

You'll start by installing Karma, which then helps set up Jasmine. Karma has a plugin for Jasmine and Chrome, which you'll install in addition to the core Karma tool. Open the command line, navigate to the project directory you set up at the start of the chapter, and run the following commands to install the tools:

```
$ npm install --save-dev karma karma-jasmine karma-chrome-launcher
$ npm install -g karma-cli
```

The first line adds Karma, the Jasmine plugin, and the Chrome plugin to the project, and saves them as development dependencies. The second line adds Karma globally so you can run it easily from the command line.

Before Karma can run, you need to add a configuration file so it knows what to do. Karma runs just the JavaScript files you specify instead of loading an HTML file and letting the page load (you'll do that in the next section). Make a new file in the project root called `karma.conf.js` and add the code from the following listing.

```
module.exports = function(config) {
  config.set({
    frameworks: ['jasmine'],
    files: [
      'www/lib/ionic/js/ionic.bundle.js',
      'www/lib/moment/moment.js',
      'www/lib/moment-timezone/builds/moment-timezone-with-data.js',
      'www/lib/suncalc/suncalc.js',
      'www/lib/angular-mocks/angular-mocks.js',
      'www/js/**/*.js',
      'www/views/**/*.js',
      'test/unit/**/*.js'
    ],
    reporters: ['progress'],
    browsers: ['Chrome']
  });
};
```

Declares you want to use Jasmine (points to `frameworks: ['jasmine']`)

Tells Karma to load files you include in app (points to the `files` array)

Adds angular-mocks file, which is used to help write tests (points to `'www/lib/angular-mocks/angular-mocks.js'`)

Uses glob patterns to match app and test files (points to `'www/js/**/*.js'` and `'test/unit/**/*.js'`)

Uses Chrome for testing (points to `browsers: ['Chrome']`)

Uses progress reporter option (points to `reporters: ['progress']`)

This configuration is what you'll use for your tests. You have to declare the framework you wish to use (in this case, Jasmine) and tell Karma which files to include. Karma will load these files into a browser (in this case, Chrome) and run all of the tests it finds. The results are then reported back to the console, but could be configured to output to a file (several file types are supported, such as HTML or XML). Now you can write a test and execute it. Any files that your app needs to run should be included in the file list, just as you've included libraries such as `Moment.js`.

WRITING A UNIT TEST FOR THE CHANCE FILTER

Jasmine is a behavior-driven development (BDD) framework. You may be familiar with different agile development methodologies; the primary idea is to help reconcile the difference between technical and management teams during the software development process. When you

write tests, you'll describe the feature with a list of statements about what it should do. I draw attention to the terms describe and it because they're used as part of the testing syntax.

Jasmine versus other testing frameworks

Jasmine is a very powerful testing framework, but it's not the only option. Several other examples are Mocha, QUnit, and Unit.js. In the world of JavaScript, new frameworks appear all the time, so you might be aware of some other new options.

In short, you should be able to use any testing framework that you desire. The more popular it is, the more likely it's well supported by the tools in this book. Jasmine is the testing framework in use by the Angular project for the 1.x version, so it's a good choice for anyone who's new to testing.

I personally enjoy Jasmine for the most part, but Mocha is another framework I've used. Jasmine provides most everything you need for testing, whereas Mocha is more piecemeal and requires you to add additional tools for certain things. Unless Jasmine is unable to meet your needs or you have more experience with another framework, I recommend it for use with Ionic and Angular apps.

RUNNING THE UNIT TESTS

To run the tests, you'll need to use the karma command-line tool. It will start a session where it will watch your files as you edit, so it can automatically run the tests any time you save a file. It will also launch a new Chrome window to run the tests inside of Chrome. Run this command from the root of your project:

```
$ karma start
```

It will start the Karma server, which watches the files and handles running the tests in the browser. It will also execute the tests immediately, and report the output of the tests directly into the command line.

Typically, I keep this command-line window open and running my tests the entire time I'm developing. It helps to remind me to write the tests, and helps me see immediately when I might have broken code.

WRITING A UNIT TEST FOR THE SEARCH CONTROLLER

Now you'll create another test for one of the controllers, as shown in listing 9.3. Most of the structure is the same, but you have to do some different setup to test a controller. You'll test the search controller, which is fairly simple, but because it makes an HTTP request, you'll have to do some mocking to test it.

Integration tests with Protractor and WebDriver

Some parts of your app are best tested with an integration test that can simulate user behaviors such as tapping or typing values into a form input. Protractor (www.protractortest.org) is a testing framework built specifically for Angular (in fact, by the Angular team), and therefore works for Ionic apps. Protractor is built on top of an API called WebDriver (<http://w3c.github.io/webdriver/webdriver-spec.html>), which allows you to programmatically

interact with an application just like the user would. WebDriver is really just the specification for how programs can programmatically interact with a browser. Selenium (<http://docs.seleniumhq.org/projects/webdriver/>) was the project that inspired the WebDriver API spec. See figure 9.10 for an example of how tests are executed using these APIs.

Protractor extends the features of WebDriver and adds better support for Angular apps. By default WebDriver runs as soon as the page is ready, but due to the Angular digest loop, your tests need to run only after Angular is ready. Protractor aids your tests by waiting for Angular to finish rendering the view before running the tests, as well as providing a few unique API calls to target parts of an Angular template. You'll see a few of these in action in the sample test.

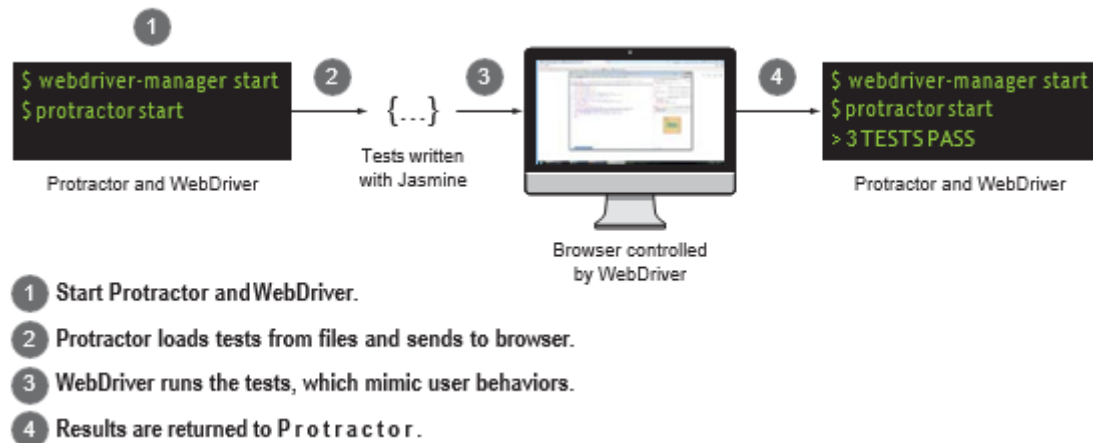


FIGURE 6.6 PROTRACTOR

During setup, you'll use Selenium (which implements the WebDriver API) and plugins for browsers (Chrome by default) to control the browser to mimic user behaviors. You'll have to have a Selenium server running in the background to run the tests, but this is easily managed by Protractor.

Protractor uses the Jasmine testing framework by default to run tests. Protractor doesn't require you to use Jasmine, so you can choose another testing framework if you'd like such as Mocha or Cucumber.js. Because you used Jasmine earlier, you can write your tests in the same style to make it a little easier.

SET UP AND RUN PROTRACTOR AND WEBDRIVER

First you need to do a one-time setup for Protractor and WebDriver. Start by installing Protractor as a global Node module, just like you did when you installed Ionic and Cordova:

```
$ npm install -g protractor
```

This command will download Protractor and also create a helper tool to easily manage WebDriver. You'll use the tool to download all of the tools for WebDriver to run. The tool will download the Selenium server and Chrome driver, and set them up:

```
$ webdriver-manager update
```

Check that everything is installed by checking Protractor's version number and the WebDriver status. You don't need IEDriver, so you can safely ignore the message about it missing:

```
$ protractor --version Version 1.6.1
```

```
$ webdriver-manager status selenium standalone is up to date chromedriver is up to date
IEDriver is not present.
```

Any time you want to run your Protractor tests, you need to first make sure the Selenium server (required by WebDriver) is running. To do this you need to have a command-line window open and run the following command:

```
$ webdriver-manager start
```

This will show a lot of diagnostic information about starting the Selenium server. This window must remain open and running any time you want to run your tests. You can stop the server by typing Ctrl-C on your keyboard. You might get a warning about Java not being installed or up to date. To fix this, download and install the latest version of the Java development kit (select the JDK option for your platform, not the JRE option) from <http://mng.bz/83Ct>.

WRITING TESTS FOR PROTRACTOR

Because you're using Jasmine, your tests will be structured similarly to the unit tests. The major difference is that you'll be focused on writing tests that mimic the behavior of the user through browser automation.

Protractor and WebDriver provide you a set of methods that you'll use to find an element on the page and interact with it. This is very similar to finding an element on the page in JavaScript using a method like `document.getElementById()`. But with Protractor and WebDriver, you can search for an element on the page by Angular-specific features, such as by the `ngModel` used on the element or the CSS class name.

Start by creating a new spec for your search view. You want to validate that the search page responds when you give it a term and press Search. Your unit tests can validate each piece works, but here you'll be validating that everything works together.

Create a file at `tests/e2e/search.spec.js` and add the contents of the following listing. These tests will use the same `describe()` and `it()` methods you saw in the unit tests.

Now that you have an idea of what the test will do, it's time to run it. You'll actually need to have three command-line windows open to run the test. The first will have ionic server running, so the website is available at `http://localhost:8100`. The second will have the Selenium server running. The third will actually run the Protractor tests. Run the following commands in separate command-line windows:

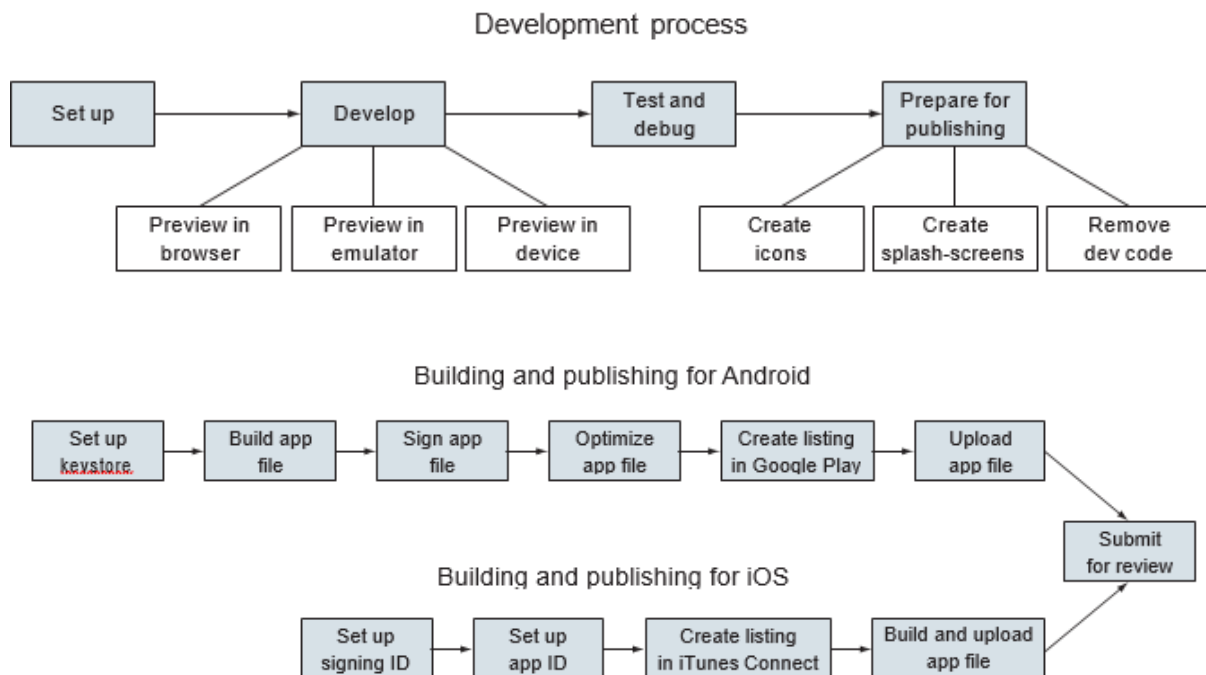
```
$ ionic serve
```

```
$ webdriver-manager start
```

```
$ protractor protractor.conf.js
```

When you run the protractor command, you'll notice that a Chrome browser will open and load the app. It will type, click, and change views very quickly; however, you should be able to actually see the interface in Chrome while the test is running.

7. BUILDING AND PUBLISHING APPS



These are the general steps for both platforms:

- You need a mechanism to sign your app for both platforms. For Android this is done with a key store, and for iOS this is called a signing identity. But they both do the same thing in the end: they add a signature to the build file that can later be used to verify the author.
- Both platforms also require you to create a listing in their store. Let's assume you've already done work to create the marketing material (screenshot images for the listing, description text, etc.), which will make it easier to create the listing. Having good marketing and app descriptions is vital for users to be able to determine if the app is for them or not.
- You must build and optimize the build file on both platforms. For Android, you upload the build file through the Google Play developer page, and for iOS, XCode connects and uploads the file to your account.

At this point, just realize the underlying steps involved are very similar for both platforms, but the nuances about how they work varies and are covered in more detail for each platform.

Building icons and splash-screen assets

As phones have improved over the years, the image quality of the graphics has needed to improve as well. To accommodate this, both Android and iOS require apps to provide a number of different sizes for icons and the loading splash-screen graphic to fit the many different screen sizes and resolutions.

For example, the iPhone 6 has a larger screen than the iPhone 5, and apps should provide a loading splash-screen image that fits both sizes of phones. Android devices also have this problem, especially because Android devices have a much greater diversity in size and resolution due to the different phone manufacturers' designs. Creating images for these

different situations can easily require dozens of images. You should also consider if you need to make a version for portrait and landscape modes, depending on the device's orientation.

Because it's somewhat painful to create so many images manually, Ionic implemented a feature that takes a single icon and a single splash-screen image and generates the various sizes that are needed for your app. It also will register the images with the cordova.xml file, so when you build the app, the images are linked correctly.

Ionic is able to convert the files by using its remote service, so your images will be uploaded to the Ionic servers for processing. This means there are no other dependencies you need besides the Ionic command-line interface. It supports PNG, PSD (Photoshop), and AI (Illustrator) files.

Creating the primary icons

To begin, you need a single icon graphic that Ionic can use to generate the rest of the sizes from. Ionic requires that you create an icon that's at least 192 pixels square, with no rounded edges. I recommend you make the icon at least 1,024 pixels square so the quality of the icon remains high. Icons are also modified slightly different for each platform; for example, iOS may round the edges of the icon. Ionic has a template for Photoshop that you can use to design your icon at <http://mng.bz/2ow0>.

There are some design considerations for the icon that you should be aware of. Both Android and iOS have some great documentation details about designing quality icons. iOS guidelines are at <http://mng.bz/B3DQ>, and Android guidelines are at <http://mng.bz/N957>. Here are a few of the top considerations:

- ☐ Keep the icon simple. Icons aren't very large, and they should be easy to see.
- ☐ Make it memorable. The icon should be something uniquely representative of your app and brand.
- ☐ Make sure it looks good large and small. Don't forget to zoom out and see if the icon still looks clear when it's small.
- ☐ Keep the colors simple. Avoid using lots of colors or colors that clash.

File locations to store icon source images

Target platform	File location
Android	resources/android/icon.png
iOS	resources/ <u>ios</u> /icon.png
Default	resources/icon.png

Any time you want to generate the icons, you just need to run the following Ionic CLI command:

```
$ ionic resources -icon
```

This may take a few moments because the files are uploaded to Ionic's servers, converted, and downloaded back into your project. Once it's complete, you should review the generated icons to confirm they appear as desired for all of the different sizes.

Creating the splash-screen images

The splash-screen works very similarly to the icon, except there's a little bit more complexity to the splash-screen design. The icons are just resized, but the splash-screen is actually resized and cropped for different resolutions and orientations. You can see in figure 10.3 how the different sizes are cropped from the source splash-screen. If you have Photoshop, you can use the Ionic splash-screen template at <http://mng.bz/2ow0> to help you design it to the correct dimensions.

The splash-screen source needs to be at least $2,208 \times 2,208$ pixels. But you should limit the custom design to a square in the center about $1,200 \times 1,200$ pixels. Typically, this inner square contains some kind of logo branding with a background color. There aren't clear guidelines for the use of splash-screens in iOS and Android, so you should consider what will provide the best experience for your users.

To generate the splash-screen images, run the following command:

```
$ ionic resources --splash
```

Like with the icons, it will upload to the Ionic servers to process the images, so you don't have to worry about having the necessary software on your machine.

If you want to generate both icons and splash-screens at once, you can just run the following command:

```
$ ionic resources
```

Preparing your app for production

There are a few things you should check to ensure your app has nothing unnecessary, which can help improve speed and stability, and reduce the app file size. You could run your automated tests to ensure that even when these steps are taken, the application still behaves as expected.

Here are some steps you should take before a release:

Remove the Cordova Console plugin. This plugin is part of how Cordova allows you to debug your apps, but in production you don't want this. Remove it from your app by running `cordova plugin rm org.apache.cordova.console`.

Remove any unnecessary files. During app development you might install extra third-party libraries or create extra views that you don't end up using. Remove them from the app so you save on file size.

Remove unused library files. Ionic may have installed files in the `www/lib` directory of your app using Bower, and sometimes those library files also include the sources. You should delete any files that you're not using.

Compress your code. You can run your code through a JavaScript minification system to help optimize the file execution and reduce file size.

Compress your graphics. Images are often what can cause app file size to grow. Try to compress your files and make sure they aren't any larger than necessary.

The main idea here is to ensure everything is ready for widespread use. You wouldn't want debugging code to appear in your app, for example. The more diligent you are about keeping your app clean while you develop, the easier this step is to complete.

7.1 Building Android apps and publishing to Google Play

Now that your app is ready to be built for production, you've got a few steps to run through to build for Android. You'll have to build the app using Cordova, sign the app to verify the source, and optimize the built app. You'll use the command line to run all of the steps for Android, but you could also read about how to use Android Studio at <http://mng.bz/T7G4>. You'll use the command-line process, as outlined in, because it's simpler for Android.

The Google Play Store is the primary place to publish your apps for Android. You'll need to create or link an existing Google account with the Play Store Developer Console. Then you'll be able to create a listing for your app that includes the title, description, images, and other details used to categorize and list the app. Once that's done, you'll upload the built Android app APK file you generated and submit the app for review.



Setting up for signing your apps

Start by setting up a keystore—a file that securely stores the security key that you'll use later to add a signature to your app. With the signature, the author of the app can be verified over time. You can read more about signing at <http://mng.bz/T7G4>.

To generate a new keystore, you'll use a command-line utility `keytool`. This generates a keystore that's valid for 10,000 days, which should be more than enough to cover the lifetime of your app (and you shouldn't make it shorter, or it might expire!). You'll replace `know_your_brew` with the name of your app (use underscores) in this command:

```
$ keytool -genkey -v -keystore know_your_brew.keystore -alias know_your_brew
-keyalg RSA -keysize 2048 -validity 10000
```

This generates a new file called, in this case, `know_your_brew.keystore`, and you can place it anywhere on your computer. Later you'll need to know the location of the file, so make sure you can access it.

You'll reuse the same keystore for the entire life of the app, so you need to keep it for as long as you plan to support the app. You also need to keep it safe and private because it could be used by others for malicious purposes. Every version of the app must be signed with the same keystore or the updates will be rejected. If a team needed to sign an app, the same keystore would need to be used regardless of who builds the app. You should also generate a different keystore for every app you produce.

Build the release app file

Next you'll build the app with Cordova. The following build command will build a release-ready version of your app:

```
$ cordova build --release android
```

This will generate a new APK file, which is the Android app file type, inside of `platforms/android/ant-build/CordovaApp-release-unsigned.apk`. The command line should report the exact file path to the APK file. This is an unsigned, release-ready version of your app.

Signing the APK file

Now you're ready to use the keystore you created earlier to sign the unsigned version of the APK you just generated. Android comes with a tool called `jarsigner` that will help you with this task.

You'll need to know the file path to both the unsigned APK and the keystore from the previous two steps. I recommend moving them into the same directory so the command is easier to type.

In the command you'll replace `know_your_brew` with the same values you used to generate the keystore for your app, and update the name of the app if it's something other than `CordovaApp-release-unsigned.apk`:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore
know_your_brew.keystore CordovaApp-release-unsigned.apk know_your_brew
```

This takes just a moment, and it will prompt you for the password for the keystore and key. It will modify the APK in place. You can test that the app is now signed properly using `jarsigner` again, and replace the name with your app filename:

```
$ jarsigner -verify -verbose -certs CordovaApp-release-unsigned.apk
```

If you have any signing errors, you might want to rebuild the app using Cordova and try again to ensure you don't have a lingering problem.

Optimize the APK

The last step is to optimize the APK file so that it reduces the amount of space and RAM required by the app on a device. The `zipalign` tool is the utility for the job: it will take your signed APK file and create a new optimized APK version that you'll want to use for uploading. Under the hood, `zipalign` will optimize the bytes inside of the package for optimal reading by the operating system processes. The technical details can be found at <http://mng.bz/vWfu>.

The `zipalign` tool just takes the name of the signed file (remember, you signed the file in place and haven't changed the filename in this example) and the name of the file to generate. Change `KnowYourBrew.apk` to the name of your app:

```
$ zipalign -v 4 CordovaApp-release-unsigned.apk KnowYourBrew.apk
```

When the new file is generated, you now have a final version of your Android app that you can use to submit to any Android store. You've finished with the initial build, but let's talk quickly about how to update your app.

Building an updated version of your app

Almost certainly you'll eventually want to update your app with new features and bug fixes. The process to build an update to an existing app is the same as building the release, except you don't need to create another keystore. A few details are worth emphasizing:

You must use the same keystore to sign the app for every update; otherwise, the update will be rejected for not having the same signature and you'll be required to create a new app listing.

You must update the version and build number in the project config.xml file for the next release. If the numbers aren't changed, then the app will not properly update on your users' devices.

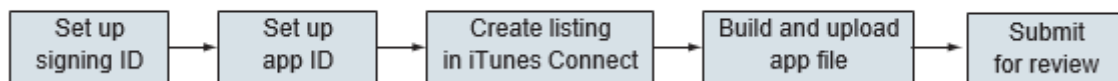
If you build frequently, you might want to improve the steps by making the commands into a shell script that can be automated.

7.2 Building iOS apps and publishing to the AppStore

To build for iOS using this process, you'll need to use a Mac and XCode, and have your Apple developer account set up for iOS development.

Apple uses iTunes Connect as the way to create a listing in the AppStore and manage the app. You'll add your app listing to iTunes Connect, fill in a lot of details such as screenshots and metadata, connect XCode to build and upload your app, and submit it for review.

Building and publishing for iOS



If you haven't set up your Apple developer account yet and registered for the iOS Developer Program, you need to do that first. Go to <https://developer.apple.com/programs/> to sign up; it costs US \$99/year to be part of the iOS Developer Program.

You can set up a new account for your apps if you have a personal account with Apple already.

Set up certificates and ID

Once you have your account, open XCode on your Mac and go to the preferences. If you haven't already added your account to XCode on the Accounts tab, do so now. This will sync XCode with your account.

Let's start with getting a signing identity (also called distribution certificate). This is used to sign an app and verify that the app was built and submitted by the account owner. You can review the official documentation about managing certificates and IDs at <http://mng.bz/64k9>.

The basic steps are these:

- 1 Log into your Apple developer account in XCode if you haven't already.
- 2 In Preferences, manage your account and certificates.

3 Create a new signing identity specifically for distribution (not development). Once the signing identity has been resolved, it should appear in the list as iOS Distribution. You may already have an iOS Development identity as well from testing.

Set up an app ID identifier

Now you'll set up the app ID identifier details through the Apple Developer Member Center. Identifiers are used to allow an app to have access to certain app services, such as Apple Pay or Health Kit. Multiple identifiers might be used in the same app for different services, but in this case you'll use just one.

Go to <https://developer.apple.com/membercenter> and log in with your Apple ID. Then choose Certificates, Identifiers, and Profiles. You want to set up a new app ID for your app, which is used to keep track of the app throughout the Apple ecosystem. See official documentation about app IDs at <http://mng.bz/8hj1>. The basic steps are these:

- 1 Start to register a new app ID.
 - 2 Supply the name of your app, and use the Explicit App ID option. Provide the bundle ID from your app, which is by default the ID in the <widget> tag you specified in the Cordova config.xml file of your app (or if you modified the bundle ID value in your XCode project). It must match your app bundle ID.
 - 3 Choose any of the services that need to be enabled. For example, if you use Health Kit in your app, you need to choose that option. Apps often have no additional services, so if you don't think you need it, just leave them as default values.
 - 4 Submit to register the app ID.
- That will take care of the ID registration for your app, and it will be used by iTunes Connect and XCode in the following steps.

Create listing in iTunes Connect

You now need to make your listing in iTunes Connect, which is the portal that Apple uses to manage app submissions. You'll use your app ID that you generated to create a new record.

Log into iTunes Connect at <https://itunesconnect.apple.com> to get started. Detailed documentation about iTunes Connect can be found at <http://mng.bz/92eZ>. The general steps are these:

- 1 Add a new iOS app.
- 2 Fill in the app details, and choose the correct bundle ID (the name of the app ID you made earlier) for the app.
- 3 Create the app listing. You'll fill out more details later.

Now you've generated a new app listing that will eventually be ready to submit to the AppStore. You've taken the app ID you created before and connected it to this app listing. Before you fill out everything in the listing, you'll build your app and get it uploaded first using XCode. Then you'll come back to finish the listing.

Build and upload app with XCode

Now that you have an app ID and iTunes Connect app listing started, XCode can help you build and upload the app. You first have to make sure that the XCode project is up to date with your Cordova project. Run the Cordova build task from the project root in the command line:

```
$ cordova build ios --release
```

This will ensure the latest changes from your project are set up in the iOS project. Open the `platforms/ios/AppName.xcodeproj` file in XCode. It should allow you to see details about your app in the general view, where you want to confirm things look correct:

The bundle identifier should match the value you specified earlier in the app ID.

The version and build numbers should reflect what you intend them to be.

Team should be set to your Apple account.

Deployment target and devices should reflect which versions and devices you intend to support.

XCode is good about prompting you to fix certain errors if you haven't set up some- thing correctly. Review any error messages, and in some cases XCode can even resolve them for you. You'll also need to make sure you don't have a device connected to the computer.

You can now build the app as an archive (which is the app bundled for uploading), and then you'll upload it. The full documentation is found at <http://mng.bz/20m2>. The general steps are these:

- 1 Create a new archive of your app, which will make a build of your app that can be later submitted.
- 2 Validate the archive you just created, which will ensure the archive can be uploaded correctly and passes validation tests.
- 3 Submit the app, which will actually submit the file to iTunes Connect.

Now that you've got your app finished and uploaded, you just need to complete the iTunes Connect listing and submit it for review!

Updating the app

To update an app, start by updating the build and version numbers. This can be done in the XCode project file, or you can update the Cordova `config.xml` file and then regenerate the iOS platform files with Cordova by removing the ios platform and adding it again.

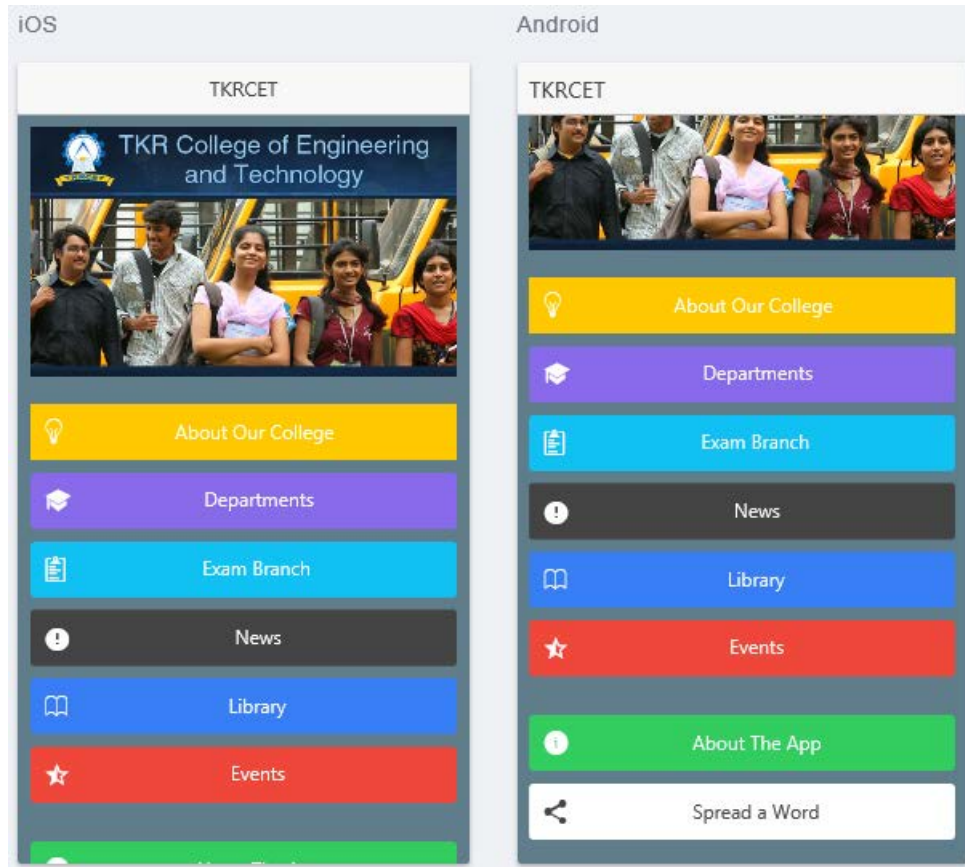
With the new version and build numbers, you can then follow the same steps to build and upload a new version to your account. If the numbers aren't updated, then the build will not upload.

Once the new package has been uploaded, you'll see a new number in the top bar for the release. Make any changes to the app listing, such as new screenshots or changing other metadata, press Save, and then press Submit for Review. The changes will go through the same review process, and the existing app will remain in place until the review is complete.

If you choose to release the version automatically, as soon as the review is completed successfully the app will go live. Otherwise, you must manually log in after the review to release a new version. Manual release might be useful if you want to trigger the release of a new version yourself at a certain time.

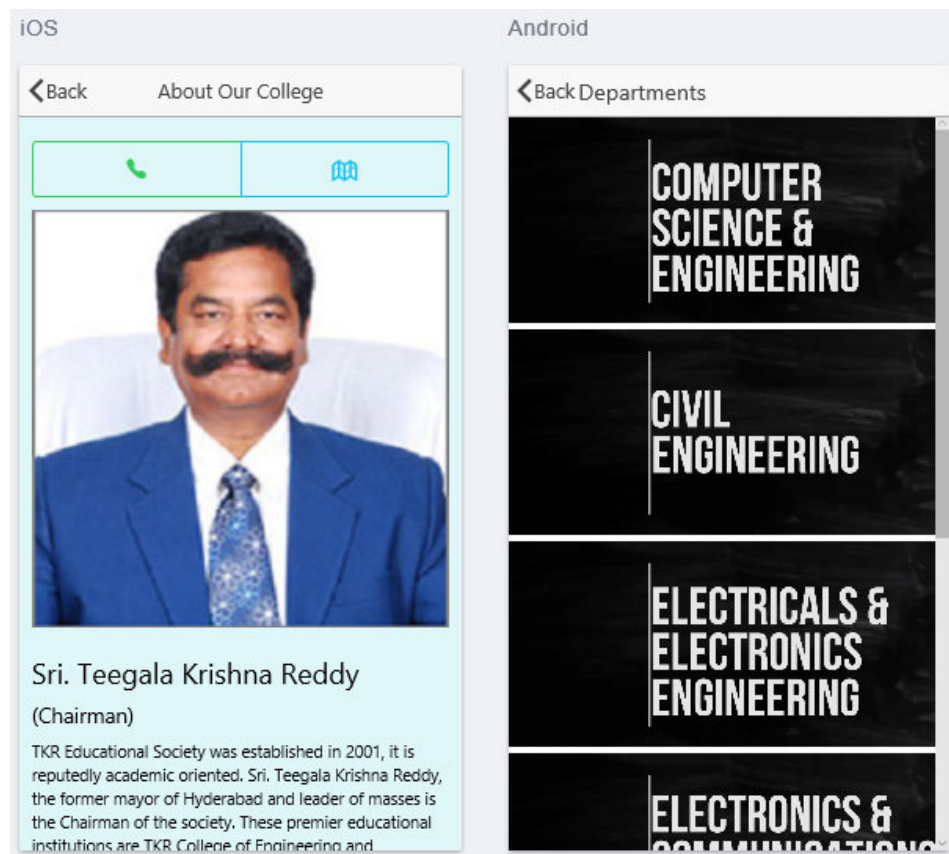
7.3 APPLICATION SCREENSHOTS

Screenshot #01



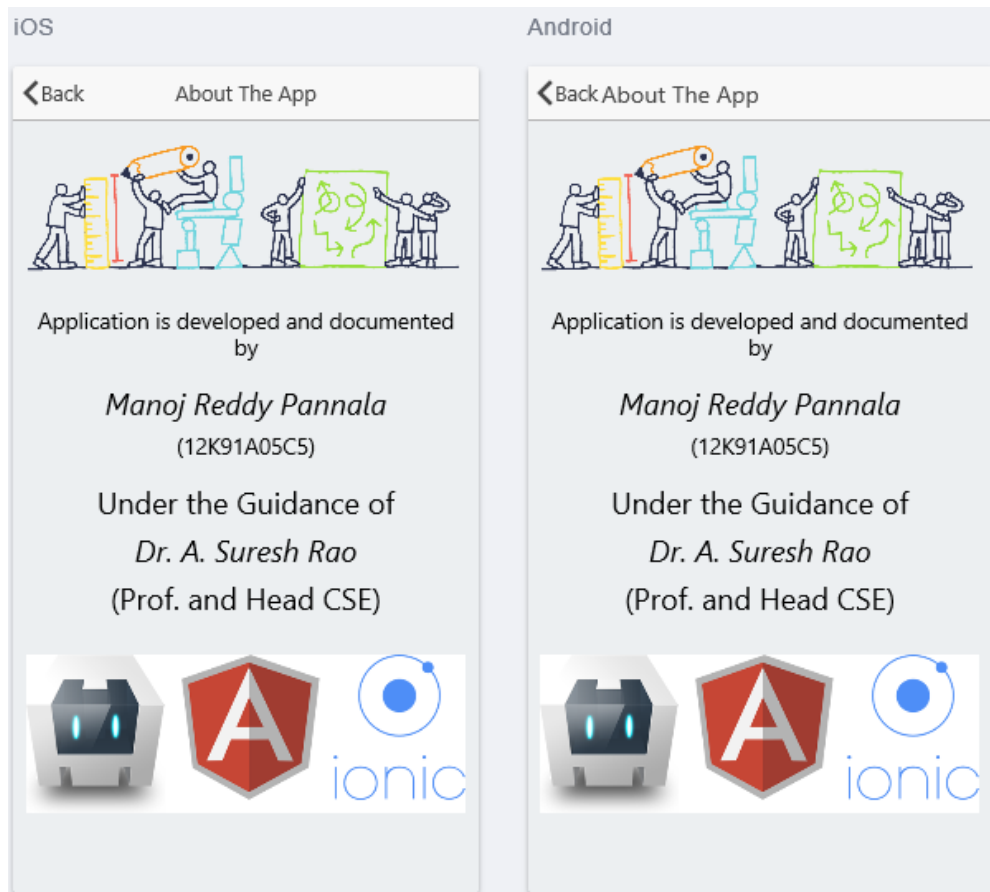
Home screenshots for both iOS and Android which contains all the buttons of its respective contents. Each button navigates to different sections of its content.

Screenshot #02



Screenshot for the About our college button and Departments button. In the Departments section, each section consists of different sub sections which navigates to its respective function.

Screenshot #03



This is the screenshot of the About the App

8. CONCLUSION

The hybrid application is much cheaper to develop, and users will not distinguish it from a native app. Hybrid is also suitable if processing speed is not important for your application.

If to compare native and HTML5 apps taking into consideration this criterion, native applications are much better. The big problem and "hole" in security is the ability to view the source code for HTML5 application. It means that you can not only understand how it works and use it, but also spam through elementary forms if there are any.

Hybrid apps successfully combine high safety and ease of development. However, there is a small nuance, which applies to HTML5-applications. With such a great variety of browsers versions in different versions of iOS and Android, a flawless performance of HTML5 and hybrid cannot be guaranteed on each unit, as some features may not be available in earlier versions of the browser.

9. REFERENCE

IONIC IN ACTION by JEREMY WILKEN

APACHE CORDOVA 3 PROGRAMMING by JOHN M. WARGO

<http://ionicframework.com>—The official Ionic website with documentation, a forum, a blog, and more.

<https://apps.ionic.io/>—The Ionic platform where you can manage your apps with Ionic View, Ionic Creator, and other Ionic platform services.

<http://ionicons.com>—A preview of all of the icons available in the Ionic icon set, Ionicons

<https://angularjs.org>—The official documentation and site for Angular 1. It contains links to starter guides, videos, mailing lists, and more.

<http://manning.com/bford>—AngularJS in Action is a complete book for getting started with Angular and learning the fundamentals.

<http://manning.com/aden>—AngularJS in Depth is a complete book about digging deeper into how Angular works, which is very useful for improving your Ionic apps.

<http://cordova.apache.org>—The official Cordova website with documentation, news, and more.

<http://plugins.cordova.io/npm/index.html>—Discover available plugins for Cordova using the official plugin registry.

10. APPENDIX

This appendix contains a curated list of additional resources. Resources shared in the documentation are also collected here as a reference.

A.1 Ionic

- <http://ionicframework.com>—The official Ionic website with documentation, a forum, a blog, and more.
- <https://apps.ionic.io/>—The Ionic platform where you can manage your apps with Ionic View, Ionic Creator, and other Ionic platform services.
- <http://ionicons.com>—A preview of all of the icons available in the Ionic icon set, Ionicons.
- <https://github.com/driftyco/ionic>—The GitHub project to follow the development of Ionic.
- <https://github.com/ionic-in-action>—The GitHub project for this book.
- <http://codepen.io/ionic/public-list/>—A list of useful demos for individual features created by the Ionic team.
- <http://mng.bz/A24v>—The YouTube channel from Ionic containing demos, tutorials, and episodes from the team.

A.2 Angular

- <https://angularjs.org>—The official documentation and site for Angular 1. It contains links to starter guides, videos, mailing lists, and more.
- <http://manning.com/bford>—*AngularJS in Action* is a complete book for getting started with Angular and learning the fundamentals.
- <http://manning.com/aden>—*AngularJS in Depth* is a complete book about digging deeper into how Angular works, which is very useful for improving your Ionic apps.
- <http://angular.github.io/protractor>—End-to-end testing for Angular is made much easier with Protractor.
- <http://karma-runner.github.io>—Karma is the popular test runner for executing unit tests built by the Angular team.
- <http://jasmine.github.io>—Jasmine is the testing library used in this book and by Angular.

A.3 Cordova

- <http://cordova.apache.org>—The official Cordova website with documentation, news, and more.
- <http://plugins.cordova.io/npm/index.html>—Discover available plugins for Cordova using the official plugin registry.
- <http://ngcordova.com>—The official ngCordova website with documentation on how to use each of the supported Cordova plugins.
- <http://manning.com/camden/>—*Apache Cordova in Action*, a great book by Raymond Camden that digs deep into the features of Cordova.

A.4 Blogs

- <http://ionicinaction.com>—The companion website and blog for this book.
- <https://blog.nraboy.com>—Nic Raboy has many good posts about building mobile apps with Ionic.
- <http://www.raymondcamden.com>—Raymond Camden blogs regularly about building mobile apps, using Cordova, and also about Ionic.
- <http://mobilewebweekly.co>—A great weekly email newsletter with carefully curated links to the top posts on mobile development from around the web.