Mastering Ionic

The Definitive Guide

James Griffiths

Mastering Ionic

James Griffiths
Saints at Play Limited
Find us on the web at: www.saintsatplay.com and www.masteringionic.com
To report errors with the book or its contents: support@masteringionic.com

Thanks to...

The team at Ionic for creating such a phenomenal product that allows millions of developers worldwide to realise their ideas quickly and easily

To every developer who ever helped me with a question that I had or a software bug that I was trying to fix - I may not remember all of your names but that doesn't mean I don't appreciate the assistance you provided at the time!

To those who believed in me and gave me a chance when many others didn't or wouldn't

Table of contents

Version History 6

Introduction 13

The last decade 22

Changes from Ionic 3 26

Core tools & technologies 40

The Ionic ecosystem 55

Configuring your environment 60

Beginning Ionic development 68

The architecture of an Ionic App 81

Decorators & Classes 90

Ionic Navigation 120

Templates 175

Theming Ionic Apps 198

Plugins 217

Loading Data 251

Forms & Data Input 267

Data Storage 291

Animations 331

Troubleshooting your Ionic App 348

Debugging & Profiling Apps 365

Documenting your code 380

Preparing apps for release 397

Code signing for iOS & Android 409

Submitting your iOS App to the Apple App Store 433

Submitting your Android App to the Google Play Store 447

Case Study #1 - Movies App 456

Case Study #2 - SoopaDoopaGames App 526

Case Study #3 - Appy Mapper App 606

In Closing 689

Author Biography 690

Index 691

Project download links 696

Further reading 698

Date	Changes
November 23rd 2016	Mastering Ionic published!
November 25th 2016	Resolved some page formatting issues Changed URL's for file downloads
December 1st 2016	Updated content to reflect changes in Ionic RC3 Added further links to file downloads
December 30th 2016	Updated forms section to cover alternate FormControl syntax in HTML field Added note to **The ionic white screen of death** section in Troubleshooting your Ionic App
February 6th 2017	• **Beginning Ionic development chapter** updated with details of the Ionic build process and Ionic serve functionality • **The architecture of an Ionic app** chapter updated to include recent file changes in final release of Ionic • **Decorators & Classes** chapter updated to fix some code typos • **Ionic Navigation** chapter updated to fix some code typos

February 6th 2017	• **Templates** chapter updated to reflect recent changes to using the Slides component API • **Forms & Data Input** chapter updated to fix some code typos • **Data Storage** chapter updated to fix some code typos • **Animations** chapter updated to fix some code typos • **Movies App** Case Study updated to fix some code typos • **SoopaDoopaGames App** Case Study updated to fix some code typos • **AppyMapper App** Case Study updated to fix some code typos
March 16th 2017	• Added examples of implementing Ahead-of-Time rendering for Ionic project builds for ALL code examples on both iOS & Android • Corrected some minor errors and omissions in the Case Study #2 chapter
March 29th 2017	• Updated information regarding how Ionic Native plugins are installed as of Ionic Native 3.x • Updated code examples to reflect changes in how plugins are installed

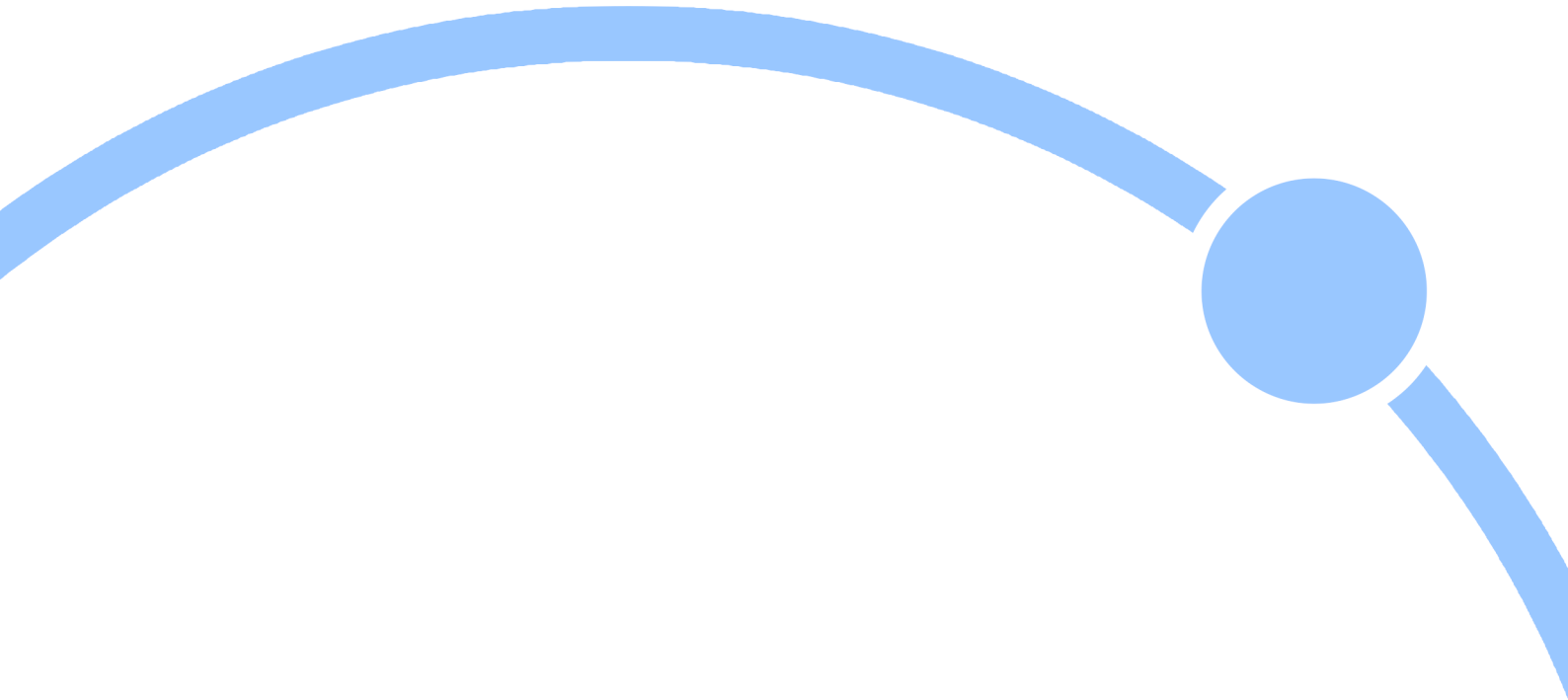
April 13th 2017	Following chapters updated to reflect recent changes with release of Ionic 3: • **Changes from Ionic 1** • **The Ionic ecosystem** • **Configuring your environment** • **Beginning Ionic development** • **The architecture of an Ionic App** • **Decorators & Classes** • **Ionic Navigation** • **Templates** • **Plugins** • **Loading Data** • **Forms & Data Input** • **Data Storage** • **Animations** • **Troubleshooting your Ionic App** • **Debugging & Profiling Apps** • **Documenting your code** • **Movies App** Case Study • **SoopaDoopaGames App** Case Study • **AppyMapper App** Case Study
April 30th 2017	Updated the **Troubleshooting your Ionic App** chapter to provide details on resolving the IonicNativePlugin error with recent releases of the Ionic Framework
May 27th 2017	Updated the following chapters to reflect recent changes to the Ionic CLI:

	• **Changes from Ionic 1** • **The Ionic ecosystem** • **Configuring your environment** • **Beginning Ionic development** • **The architecture of an Ionic App** • **Decorators & Classes** • **Ionic Navigation** • **Templates** • **Plugins** • **Loading Data** • **Forms & Data Input** • **Data Storage** • **Animations** • **Troubleshooting your Ionic App** • **Debugging & Profiling Apps** • **Documenting your code** • **Movies App** Case Study • **SoopaDoopaGames App** Case Study • **AppyMapper App** Case Study
July 10th 2017	Additional Ionic CLI commands added to the **Beginning Ionic Development** chapter
July 17th 2017	Updated the following chapters to reflect recent changes with the release of Ionic 3.5: • **The architecture of an Ionic app** • **Animations** • **Case Study #1 - Movies App**

July 25th 2017	Updated the following chapters to reflect changes with the release of Ionic 3.5: • **The architecture of an Ionic App** • **Decorators & Classes** • **Ionic Navigation** • **Templates** • **Plugins** • **Forms & Data Input** • **Data Storage** • **Animations** • **Debugging & Profiling Apps** • **Documenting your code** • **Case Study #1 - Movies App** • **Case Study #2 - SoopaDoopaGames App** • **Case Study #3 - AppyMapper App**
July 28th 2017	Updated the following chapters to reflect changes with the release of Ionic 3.6: • **Beginning Ionic development** • **Decorators & Classes** • **Case Study #3 - Appy Mapper**
November 8th 2017	Update the following chapters to reflect changes with release of Ionic 3.17.0: • **Beginning Ionic development** • **The architecture of an Ionic App**

November 28th 2017	Updated the following chapters to reflect changes with release of Ionic 3.19.0: • **Changes from Ionic 1** • **Beginning Ionic development** • **The architecture of an Ionic App** • **Ionic Navigation** • **Templates** • **Theming Ionic Apps** • **Plugins** • **Loading Data** • **Forms & Data Input** • **Data Storage** • **Animations** • **Debugging & Profiling Apps** • **Documenting your code** • **Case Study #1 - Movies App** • **Case Study #2 - SoopaDoopaGames App** • **Case Study #3 - AppyMapper App**
February 18th 2018	Updated the following chapters to include details of recent developments with Ionic: • **Introduction** • **Changes from Ionic 1** • **Core tools & technologies** • **The Ionic ecosystem**

May 26th 2018	Fixed formatting issues with page numbers and updated text on publications details on page 660.
November 4th 2018	Updated the following chapters to cover changes with Ionic 4 beta release: • **Changes from previous versions of Ionic** • **Core tools & technologies** • **The Ionic ecosystem** • **The architecture of an Ionic App** • **Decorators & Classes** • **Ionic Navigation** • **Templates** • **Plugins** • **Forms & Data Input** • **Data Storage** • **Animations** • **Debugging & Profiling Apps** • **Documenting your code** • **Case Study #1 - Movies App** • **Case Study #2 - SoopaDoopaGames App** • **Case Study #3 - AppyMapper App**



Introduction

Thank you for purchasing this copy of Mastering Ionic.

My goal with this book is to take you through using the Ionic framework; beginning with an understanding of the individual products and services of the Ionic ecosystem and their underlying technologies before moving on to the different features of Ionic.

Later chapters of the book guide you through creating fully fledged mobile apps and understanding how to submit those to both the Apple App and Google Play stores.

If you are familiar with using Ionic 1 you might be surprised, and possibly feel a little overwhelmed, at the changes to the framework. If this is the case, or you're maybe a little hesitant at moving straight into Ionic development, I'll take you through those changes, helping you to not only understand them but also familiarise yourself and become comfortable working with the new syntax - I promise that before you know it you'll be hitting the ground running!

Please be aware that this book will not cover Ionic 1 development, unless mentioned in passing, and the development focus will be solely from a Mac OS X perspective - you will, however, be able to use all of the non Xcode related coding examples regardless of what platform you use.

Prerequisites
You should already understand and be familiar with using HTML5, CSS3, Sass, Angular JS and JavaScript in your projects as this book will not teach you how to use those languages other than providing examples for you to build on your existing knowledge with.

If you are not familiar with any of these languages I recommend you start with the following resources before reading this book:

- HTML5 and CSS3 for the real world
- Beginning JavaScript 5th Edition
- Jump Start Sass

Although this isn't mandatory you should already have some familiarity with using the command line interface (CLI). If you're new to this don't worry - it's very quick and easy to learn from the examples that I take you through.

You should also be familiar with object-oriented programming (OOP) concepts such as classes, inheritance and access modifiers.

I will explain and elaborate on these concepts in the context of the Ionic framework but it will help greatly if you already have some prior knowledge of and experience working with object-oriented programming.

If you don't I would recommend the following resource to start with: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Object-oriented

So why choose Ionic?
Out of the numerous other mobile development frameworks that are available why should you choose to develop with Ionic?

In my humble opinion Ionic offers the best feature set and related product/service ecosystem of any mobile development framework on the market today.

Built to work seamlessly with both Apache Cordova and Angular, Ionic brings the features and functionality of those frameworks while providing a generous library of pre-built components and device plugins to choose from.

All of which are based on modern standards and able to be built on top of by other developers (and I'll cover all of these points throughout the book so you have a thorough understanding of what Ionic is from the ground up).

Oh, and in addition to these technologies Ionic is free to download and use in your projects.

Need I say more?

My background

My first introduction to cross-platform mobile development frameworks started with jqTouch back in mid-2010 followed by jQuery Mobile and then a brief flirtation with Sencha Touch before finally settling on Ionic Framework in August 2014 when it was nearing version 1 of the SDK.

Since then I've used Ionic to develop a number of mobile/tablet apps for both iOS and Android on client projects as well as internal projects for my own company, Saints at Play Limited.

So what made me choose Ionic over other frameworks that I've used (or could have used instead)?

Well, where to start?

With Ionic I get access to the plug-in/device API architecture of Apache Cordova (courtesy of Ionic Native), in addition to being able to publish my app to different mobile platforms, as well as the modularity, rich event handling and data binding capabilities of Angular.

Then there's the considerable library of pre-built UI components, command line tools, support for modern web technologies, extensive online API documentation of the framework itself and, if all else fails, the well managed and helpful support forums to find answers to those seemingly unanswerable questions that often plague us as developers.

All of these, in particular the seamless integration with Apache Cordova and Angular, makes developing mobile apps with the Ionic framework not only easier but also quicker - and a lot more fun too!

And, for the purposes of disclosure, I am NOT affiliated with either the Angular or Ionic teams nor am I sponsored to promote their products in any way, shape or form whatsoever - I'm just a developer who likes using their tools.

Just so you know!

Conventions used within this book

You'll be pleased to know that there's only a small number of conventions used in this book (which means you can spend more time reading and learning than trying to remember what such and such a convention is used for!)

Where code examples are listed these are always displayed within grey rounded rectangles, and may (or may not) contain comments for further context, like so:

```
// Installing the Ionic CLI on Mac OS X
sudo npm install -g ionic cordova
```

Important information covered within each chapter is prefixed with the capitalised word important like so:

IMPORTANT - pay attention here

Code that has been previously covered will, where further additions to that example are being shown, be displayed using a placeholder in the following format:

...

Code may sometimes run onto a second line with a hyphen inserted into the code to show that the first line is connected with the second line. Such hyphens do NOT form part of the code.

Summaries are provided at the end of most chapters to help quickly consolidate key concepts and topics covered.

Resources for further learning are also listed as hyperlinks, if and where necessary, throughout each chapter to help the reader build upon what they learn at each stage of this book.

Support

To accompany this e-book I've developed a website: http://masteringionic.com which

contains the following resources:

- Articles covering aspects of developing with Ionic
- Lists any book errors that have been found
- Contains downloads for code covered in this e-book (you WILL need these!)
- Support forms where you can contact me directly
- Links to further e-books
- Mailing list subscription forms (please sign-up to the list as you'll receive e-book updates as well as news/discounts on further e-books that I'll be publishing)

Please feel free to drop me a line and get in touch; whether it's potential issues with the e-book (especially if something doesn't work or appears inaccurate), sharing your thoughts, suggestions for content or just to say hello, I'd love to hear from you!

Links to the project code samples for this book are listed on page 696.

IMPORTANT - IF you should experience issues or errors with the code for featured projects/applications not working (I.e. such as build failures or content not rendering) DO read through the *Troubleshooting your Ionic App* chapter for potential fixes and solutions.

This might save you much head scratching, confusion, swearing and time spent looking for anwers - if nothing else works DO get in touch with me and I'll help where I can!

Update schedule

With new releases of the Ionic Framework I aim to update the e-book content and related code (where required with breaking changes and/or new developments) within 4 - 7 days of said releases.

One final notice: currently this e-book focuses solely on developing mobile apps with Ionic - in a forthcoming iteration I'll also introduce chapters on using Ionic to create Progressive Web Apps - so stay tuned for updates!

Technical Terms
Before we complete our introduction there are some programming related terms that will be popping up throughout the book that, to avoid any confusion, should best be explained here.

If you're already familiar with these terms, and I imagine most of you reading this book will be, then press on to the next chapter! If not, please take a moment to read through the following terms and their definitions:

Bootstrapping
A term used in programming to describe the first piece of code that runs when an application starts and is responsible for loading the remainder of the application code

Class based language
A style of object-oriented programming where objects are generated through the use of classes

CLI
Short for Command Line Interface which is a means of interacting with a program through inputting text commands on the screen

DOM
Short for Document Object Model - a cross platform API that implements a tree-like structure to represent HTML, XHTML and XML documents

ECMAScript
The language standard that governs the core features of JavaScript (which is a superset of ECMAScript)

ES6
Short for ECMAScript 6 (also known as ECMAScript 2015) which is the next version of the JavaScript language that introduces class based object oriented programming features to the language

Hybrid Apps
These are applications developed using web based languages such as HTML, CSS and JavaScript and then published within a native wrapper so they are able to run on different mobile platforms such as iOS & Android

JSON
Short for JavaScript Object Notation - a language independent data format based on a subset of JavaScript

Keychain Access
An Apple software utility that lists all the passwords, private/public keys and digital certificates that you have previously generated and stored

Native Apps
Applications that have been developed to be used only on a specific platform or device and are often able to interact with the operating system/software of that specific platform

Node
An open-source, cross platform JavaScript environment for developing server-side web applications

Object-oriented programming
An approach to programming organised around the concept of objects

OS
Short for operating System

Package Manager
A tool, or collection of tools, for managing the installation, configuration, upgrading, removal and, in some cases, browsing of software modules on a user’s computer

Progressive Web Apps
Progressive Web Apps, often abbreviated to PWAs, are mobile applications designed to leverage modern web features to deliver Native App-like experiences to

the end user while running in the browser

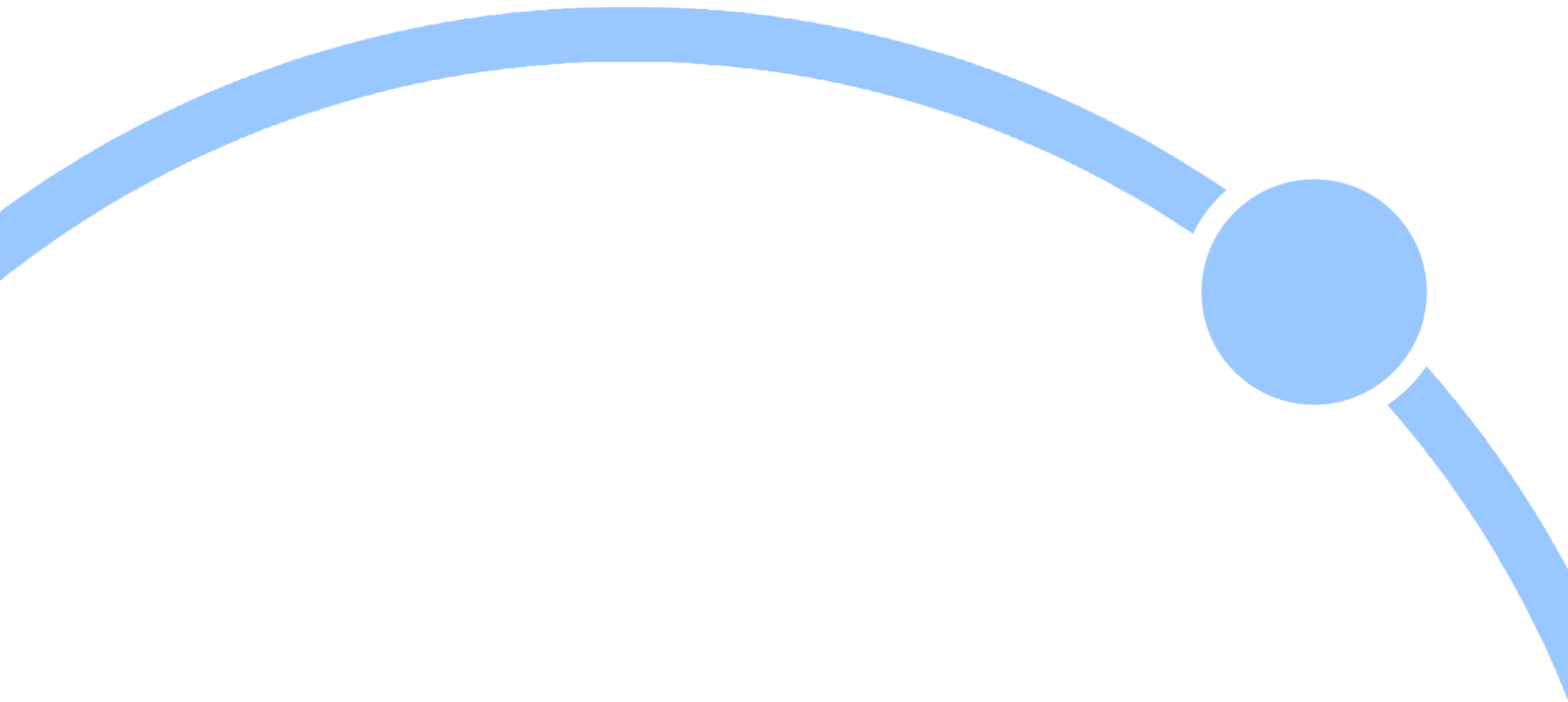
Prototype based language
A style of object-oriented programming that, instead of using classes, relies on the cloning of objects, with each generated object inheriting behaviour from their parent

Superset
A programming language that contains all of the features of its parent or associated language but implements additional features not found within the parent language

Version control
Both software tools for and an approach to the management of changes to files and applications over time

Virtual machine
A software representation of a physical computer running an operating system and applications on the same device as the native operating system. For example, a virtual machine can allow a host OS such as Mac OS X to run a Windows OS or Linux OS on the same computer

Xcode
An integrated development environment developed and maintained by Apple that is used to create, test, compile and publish Mac OS X & iOS applications



The last decade

It’s been an interesting journey.

As mobile app developers we’ve come a long way since the introduction of the first iPhone at the Macworld Conference & Expo in January 2007, followed by the first commercial release of Android in September 2008.

Following from both of these global interest in smartphones exploded and public adoption quickly followed but there was one major stumbling block for developers...

Unless you used platform specific APIs you couldn’t join the party.

This meant you had to learn, if you weren’t already familiar with, Objective-C (iOS), Java (Android) and C# (Windows Mobile Apps) if you wanted to develop apps for these popular mobile platforms (despite Windows Mobile rating a very distant third compared to iOS & Android).

Unfortunately this issue of platform specific APIs also brought with it the cost, complications and headaches of having to plan, create, debug, publish, maintain and coordinate separate codebases for the same app for each mobile platform being targeted.

Predictably, and understandably, this was not the most attractive proposition for any development team or client looking to manage their time, resources and, ultimately, expenditure.

In 2009 an iPhoneDevCamp event in San Francisco was to provide an opportunity for two developers to start exploring an idea that would change this approach.
Rob Ellis and Brock Whitten began work on a development framework that would allow web developers to create mobile apps through simply packaging HTML5, CSS & JavaScript code that they had written into native containers that could then be published for specific mobile platforms.

This framework would eventually morph into the free, open-source Apache Cordova project - as well as the separate but similar Adobe PhoneGap service.

Now there was no need for a developer to have to learn Objective-C, Java or C# in order to develop mobile apps. They could simply stick with the familiar day-to-day web technologies that they used and package these into native containers instead.

As if this wasn't groundbreaking enough developers could now use one codebase to publish to different platforms with. Opportunities for mobile app development were finally accessible to web developers without the burden of having to learn multiple different languages.

Thanks to this innovation hybrid apps were not only a reality but were now able to be created at a fraction of the cost and time of "traditional" native apps.

Aside from the cost and time benefits of not having to create and maintain a number of separate codebases for the same app, these hybrid app development frameworks also provided developers with one very important tool: a plugin architecture offering a simple but powerful JavaScript API.

This allowed device functionality such as geolocation, capturing images using the camera hardware or accessing address book contacts to be integrated directly through JavaScript - giving hybrid apps a tremendous amount of parity with their native counterparts (although, at the time, there were still some notable differences in performance, particularly with intensive gaming applications).

And the aesthetics/user experience - such as implementing the smooth transition from one screen view to the next or providing a standardised, user friendly UI?

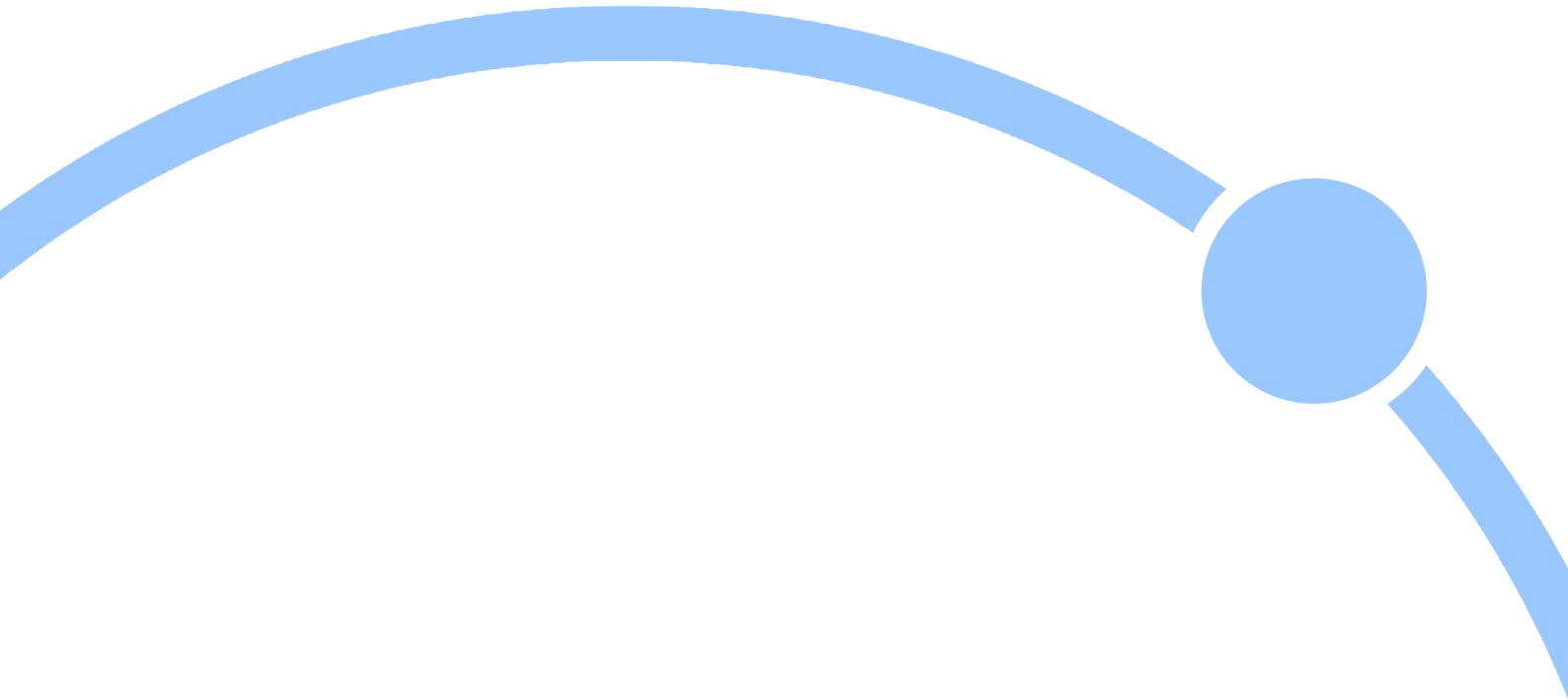
These could now be handled using different front-end development frameworks such as Onson UI, Framework 7, jQuery Mobile, Sencha Touch, Kendo UI or Ionic Framework.

And so it is, within the space of a decade since the release of the first iPhone, that app development has evolved, and continues to evolve, at such a rapid pace with no signs of slowing down any time soon.

As a developer you now have the flexibility and freedom to be able to code for iOS,

Android, Windows or all 3 platforms using a set of, mostly free, open-source tools without having to learn a variety of programming languages in the process - something that was impossible to accomplish back in the nascent days of mobile app development.

Ionic is just one of these open-source tools and it’s this particular app development framework that we will focus our attention on over the remaining chapters of this book...



Changes from previous versions of Ionic

Compared to previous versions of the framework the latest Ionic release brings with it a number of feature changes and enhancements which include:

- Web Components
- Greater Angular integration (particularly with navigation)
- Framework agnostic

As well as pre-existing features such as the following (which those developers new to Ionic/Angular may not be familiar with - so we'll cover these too):

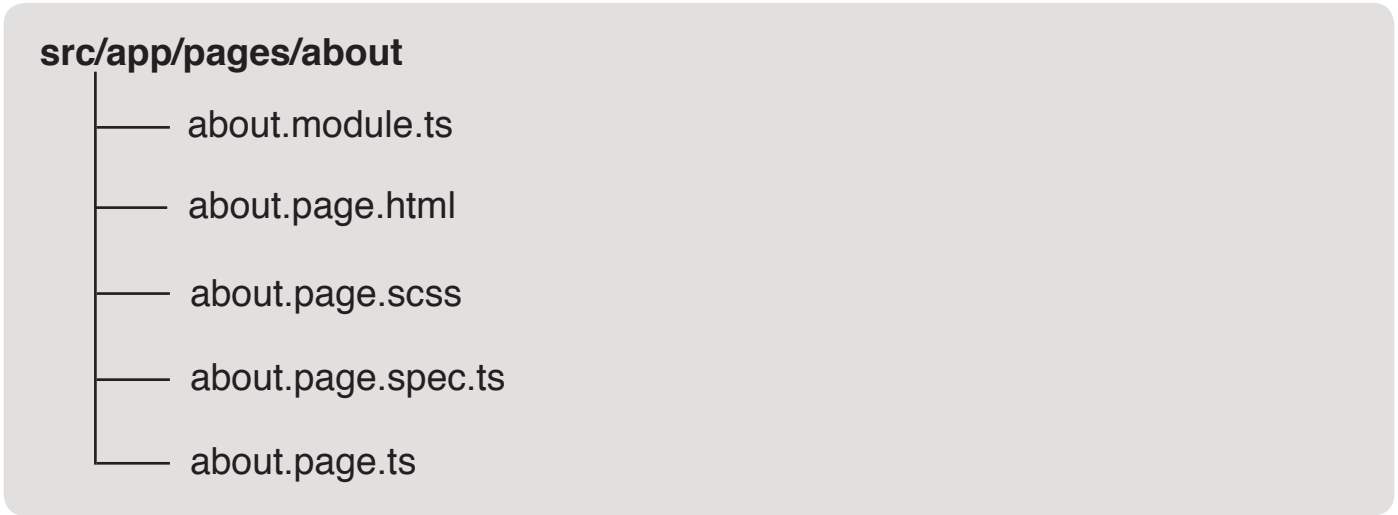
- TypeScript
- Native support for Promises and Observables
- Arrow functions
- Ahead-of-time compilation

IMPORTANT - Throughout the remainder of the book (unless otherwise indicated) we will be using the Angular framework to explore Ionic application development.

Web Components

Web components are self contained packages of HTML, TypeScript and CSS/Sass which help form the individual pages for our Ionic apps.

For example, an About page component is simply a directory named after the page with the following file structure:



These files provide the following functionality:

- **about.module.ts** - Angular feature module for loading components, directives, pipes and services applicable to that component (don't worry if these terms don't make sense right now - we'll cover them in subsequent chapters!)
- **about.page.html** - The HTML template which will be rendered to the screen
- **about.page.scss** - The CSS styles applicable to only that page component
- **about.page.spec.ts** - Stores any unit tests that will be written for the component
- **about.page.ts** - Stores the logic for managing the page component

This means that developers not only get the page logic, styling and templating for that component but can also determine which packages and modules are imported solely for that page component (thanks to the **module.ts** file) as well as being able to write self-contained unit tests (thanks to the **spec.ts** file).

As of Ionic 4 all components (whether page components that are generated via the Ionic CLI - such as the previous example and those which we'll cover in subsequent chapters - or pre-existing UI components such as cards, lists and buttons) are built using technologies based on the Web Components API.

These technologies include:

- **Custom Elements** - A set of JavaScript APIs that are used to create new HTML elements and define their behaviour
- **Shadow DOM** - JavaScript APIs for creating a shadow DOM "tree" which allows a feature to be separate from the main document DOM and not "interfere" with other parts of the document
- **HTML Imports** - Allows for HTML elements to be imported and subsequently reused in other documents
- **HTML Templates** - <template> and <slot> elements allow reusable chunks of HTML to be defined which are then able used as part of a custom elements structure

The team at Ionic have developed their own compiler, named StencilJS, to author web components which make use of features such as virtual DOM, Asynchronous

Rendering, JSX Rendering and Reactive data binding.

We'll be exploring StencilJS in a later chapter.

TypeScript

TypeScript is a superset of JavaScript (I.e.it contains all of the features of JavaScript as well as additional features not found within the language) and provides class based object oriented programming and static typing.

We'll be covering TypeScript in more detail in subsequent chapters - particularly for those developers not familiar with the language - but we'll quickly examine a sample TypeScript file (which is easily identified with the file suffix **.ts**) below:

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

@Component({
  selector: 'app-about',
  templateUrl: './about.page.html',
  styleUrls: ['./about.page.scss'],
})
export class AboutPage implements OnInit {

  constructor() { }

  ngOnInit() {

  }

}
```

Even though this is a very basic example there's quite a few things going on here:

- All modules used in the page are imported using the **import** command
- The **@Component** decorator provides metadata about the class such as which view template to use (we'll cover decorators and templates in later chapters)

- A class named **AboutPage** which is able to be imported as a module into other pages thanks to the **export** declaration
- The constructor for our class allows modules to be injected for initialisation - we'll cover dependency injection a little later on)
- Angular's **ngOnInit** lifecycle event (which allows developers to place code within this method to be triggered when the component template has been loaded)

We'll be covering components in more depth throughout the remainder of this book.

Promises

Promises are essentially an agreement for how asynchronous operations, such as managing ajax calls or retrieving records from an SQLite database, will handle data that is expected but hasn't been returned to the application yet.

Traditionally JavaScript developers would have used callbacks to handle these but Promises provide a much more efficient way of handling the success or failure of such an operation.

The following example demonstrates how a Promise can be created using the **new Promise()** constructor along with the use of the resolve and reject functions:

```
var promise = new Promise(function(resolve, reject)
{
  // Perform a task here, probably asynchronous

  if (/* task was successful */)
  {
    resolve("Everything worked");
  }
  else
  {
    reject(Error("Big time fail!"));
  }
});
```

Results from the Promise can then be accessed like so:

```
promise.then(() =>
{
   // Handle the result here
})
.catch(() =>
{
   // Handle error situation here
});
```

The **then()** method returns the results of the Promise object which can then be handled and manipulated by the script as required.

If the Promise is rejected this is able to be handled using the **catch()** method.

We'll be making regular use of Promises in code samples that are used in various chapters throughout the remainder of this book and further resources on managing asynchronous tasks with Promises are available below:

Mozilla Developer Network - the Promise object
Promise then method
Promise catch method
Promise resolve method
Promise reject method

Observables
Introduced with Angular 2 Observables, similar to Promises, are also used to handle asynchronous operations but, unlike Promises, provide the following advantages:

- Allows for the detection of real-time changes
- Can handle multiple values over time
- Supports array operators such as filter, map and reduce

- Are able to be cancelled

This would make features like live updates, such as those found in a Twitter feed for example, a perfect case use for Observables.

So what might an Observable look like?

```
nameOfService.load().subscribe(
  (data) =>
  {
    console.dir(data);
  },
  error =>
  {
    console.log(error);
  });
```

Doesn't look all that different from how a Promise handles returned data does it?

The important difference though is the use of the **subscribe()** method which allows for events to be "listened" to and the detection of changes in real-time.

Ionic implements Observables through Angular's use of the Reactive Extensions for JavaScript library, otherwise known as RxJS.

We'll be looking at Observables in action in subsequent chapters.

Reactive Extensions library
Introducing the Observable
RxJS and Observables

Angular routing and navigation
In Ionic 3 you would have relied on the default NavController/NavParams modules for handling navigation logic within your app, like so:

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

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

  constructor(public navCtrl: NavController) { }

  viewAccessory(accessory)
  {
     this.navCtrl.push('AccessoryDetail', accessory.id);
  }

}
```

How the above TypeScript file implements navigation is as follows:

- Imports the **NavController** component (for setting navigation items)
- In the class constructor we inject our imported component as a dependency to be used by our script - assigning that to a public property named **navCtrl**
- A function called **viewAccessory**, which can be called via a click event in the page template, is used to implement the actual page navigation
- This function, using the **navCtrl** property as a reference, implements the push method of the **NavController** component to state which page to navigate to (this would be in the form of a string) and what parameters will be passed to this page

And the page that we want to receive the passed in data to?

That script might look something like this (key aspects highlighted in bold):

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

@IonicPage()
@Component({
  selector: 'page-accessory-detail',
  templateUrl: 'accessory-detail.html'
})
export class AccessoryDetailPage
{
  constructor(public navCtrl: NavController,
                    public navParams: NavParams)
  {
    console.dir(this.navParams.data);
  }

}
```

So what our above TypeScript file does is:

- Import the **NavParams** component (for managing received navigation data that was set in our previous TypeScript file by the **NavController** component)
- In the class constructor we inject our imported components as dependencies to be used by our script making reference to the **NavParams** component through the **navParams** property
- Render the passed data from the **navParams** property to the browser console

This was the Ionic 3 way of handling navigation within your applications which is now - as of Ionic 4 onwards - considered deprecated in favour of Angular routing instead (you can still choose to use the NavController/NavParams approach within your applications but its use is discouraged).

If we were then to recreate our previous example using Angular routing - which we'll cover in more depth within the Navigation chaper - our code might look as follows:

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

@Component({
  selector: 'app-accessories',
  templateUrl: 'accessories.page.html',
  styleUrls: ['accessories.page.scss'],
})
export class AccessoriesPage {

   constructor(private _router      : Router)
   {  }

  viewAccessory(accessory)
  {
     this._router.navigateByUrl('/accessory' + accessory.id);
  }

}
```

Here we simply import the Angular Router module at the top of the page component, make use of its **navigateByUrl** method to instruct the application that we want to navigate to the accessory page and send a URL parameter of accessory (which could contain any type of value we want the recipient page to receive).

Following from this our **src/app/app-routing.module.ts** file (where our application navigation routes will be defined) might contain the following routes configuration:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: './home/home.module#HomePageModule' },
```

```
  { path: 'accessory', loadChildren: './pages/accessory/ accessory.module#Ac-
cessoryPageModule' },
  { path: 'accessory-detail/:id', loadChildren: './pages/accessory-detail/ ac-
cessory-detail.module#AccessoryDetailPageModule' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

What we have here are 4 separate routes:

- An empty path (which redirects to the home path)
- The home path
- The accessory path
- The accessory detail path (which supplies a URL parameter of id and defines the target destination to navigate to)

With the exception of the empty path route definition each route makes use of the Angular router's **loadChildren** property to lazy load the specified feature module for that path (I.e. the **AccessoryDetailPageModule**).

As this targeted feature module subsequently imports, defines the route for and declares the page component for that path the page is then navigated to and rendered to the device screen.

And the page component that we want to receive the passed in data to?

That page component might look something like the following (key aspects of the script are highlighted in bold):

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-accessory-detail',
  templateUrl: './accessory-detail.page.html',
  styleUrls: ['./accessory-detail.page.scss'],
})
export class AccessoryDetailPage implements OnInit {

  constructor(private _route: ActivatedRoute)
  {
     console.log(this._route.snapshot.paramMap.get('id'));
   }

}
```

Here we import the **ActivatedRoute** module from the Angular Router package and use the **snapshot.paramMap.get()** method to obtain the value of the supplied URL parameter (which we subsequently log to the device console).

Once again, we will be exploring Angular routing in the **Navigation** chapter so don't worry if the previous example might appear a little convoluted and/or confusing right now - I promise it will all make sense by the end of the book!

Arrow functions

TypeScript (as well as the latest version of JavaScript, which is guided by the ECMAScript 6, or ES6 for short, language specification) supports a feature known as Arrow Functions (often referred to as fat arrow syntax):

```
() => {
  // logic executed inside here
}
```

Arrow functions may look a little odd at first but they are, essentially, a simplified way of writing function expressions.

We'll be making use of these throughout the code samples displayed over following chapters so don't worry if they feel a little "uncomfortable" at first.

ES6 Arrow functions

Ahead-of-Time compilation
The underlying Angular framework compiles apps using one of two approaches:

- **Just-in-Time (JiT)** - Executed at runtime (in the browser)
- **Ahead-of-Time (AoT)** - Executed once at build time

Ionic implements AoT to pre-compile templates during the build process instead of during the browser runtime or on the fly. Doing so helps to catch TypeScript errors as well as increase overall performance through decreasing app boot-up times and optimising the code to run faster.

Ahead-of-Time compilation
Ionic AoT
Angular Ahead-of-Time compilation

Framework Agnostic
Although Angular is still the default front-end framework, as of Ionic 4, the CLI tool provides support for development with the following alternatives (that is at the current time of writing - August 2018 - although this list may grow over the coming months to encompass additional frameworks):

- React
- VueJS
- Preact

Or, if you prefer going it alone, NO framework at all.

This is a MASSIVE change from previous versions of Ionic Framework as it allows developers to use Ionic with any (or no) framework of their choice.

As Ionic's component architecture is now built using StencilJS any changes to the framework are completely divorced from those of a third-party framework (such as VueJS, Angular or React for example) making for a smoother transition between updates to Ionic and those of third-party frameworks.

As you can imagine this framework decoupling approach results in a HUGE win-win situation for developers and organisations looking to use Ionic within their projects.

Simply drop in your current front-end development framework of choice and start developing!

We've only really scratched the surface with this overview of changes and features within the latest version of Ionic but I'm sure you've started to get the feel for how useful they will be as part of your application development toolkit.

Over subsequent chapters we'll explore these in more detail but, before we do, let's take a look at the underlying core technologies used within Ionic...

End of sample preview...