

Flex Mobile

IN ACTION

Jonathan Campos



MEAP

 MANNING



**MEAP Edition
Manning Early Access Program
Flex Mobile in Action MEAP Final Version**

Copyright 2012 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Part 1 Getting Started

1. *Getting to Know Flex Mobile*

Part 2 Mobile Development with Flex

2. *Get Going with Flex Mobile*
3. *Persisting Data*
4. *Use Your Device's Native Capabilities*
5. *Handle Multi-Resolution Devices*

Part 3 Advanced Mobile Development

6. *MVC with Mobile Applications*
7. *Architect Our Application for Multi-Screen Development*
8. *Extending Our Mobile Application*
9. *Effective Unit Testing*

Part 4 Deployment

10. *The Almighty Application Descriptor*
11. *Building Our Application with Flash Builder*
12. *Automated Builds Using Ant*

1

Getting to Know Flex Mobile

In this chapter:

- Defining Multi-Device & Multi-Screen
- The Great Debate: Native vs Cross-Platform
- Mobile Components
- Hello World Example

You, a mild mannered programmer working tirelessly on your computer to create desktop and web applications for your own personal gain and clients, suddenly get a call on your phone from a new client asking for a mobile application. You spring into action and take the case only to realize that later that mobile development is very different from the application development you've done in the past. Enter Flex Mobile.

With the latest release of Flex, Flex 4.6 – the successor to Flex 4.5 (codenamed Flex Hero), we can easily make applications that run on the web, desktop, or wide range of mobile phones and tablets with the same codebase and familiar development techniques between each platform. This means you don't have to learn a new language or relearn how a button works each time you need to need to deploy to a new device. Instead we will use the knowledge we have, extend it, and deploy to each new platform.

What makes Flex so wonderful for application development? Built into the Flex framework is a myriad of components created with the soul purpose to create amazing applications. With many tested, extendable components we can create application easily without having to recreate the "wheel" from scratch each time.

In this chapter we will evaluate the latest changes in mobile application development, how to decide between going native and using a cross-platform solution. Once we see the benefits of the Flash Platform for Multi-Screen development we will look at what is Flex Mobile, a basic Hello World example, and finally introduce the running example that will be used throughout the entirety of the book.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=796>

1.1 *Learning Key Terms*

Before jumping right into Flex Mobile it is important to understand some key terms surrounding mobile and discuss the debate between native and cross-platform development.

NATIVE DEVELOPMENT

When we talk about “going native” we are saying to use the device’s native software development kit (SDK), and therefore programming language, to create our applications. For iOS development you’ve probably heard that native development means learning the Objective-C language while Android includes its own framework built around the Java language.

CROSS-PLATFORM DEVELOPMENT

When we talk about “going cross-platform” we are talking about using one of the development platforms, such as Flex (ActionScript) or HTML (JavaScript), to create application that work similarly, if not the same, across all devices.

MULTI-DEVICE AND MULTI-SCREEN

Thanks, in part, to the changing mobile landscape two new terms are thrown around surprisingly often, multi-device application development and multi-screen application development. While some circles may argue the unique differences to these terms, on the whole these terms are synonymous.

When we talking about multi-device or multi-screen application development we are ultimately discussing creating a single application that works on multiple devices or multiple screens. Depending on your interpretation of the word you may believe that the code is 100% the same between each device or that the application’s need to share a common codebase.

For some developers when we talk about multi-device we are discussing the various mobile platforms and when we talk about multi-screen we are also bringing in web, desktop, and television “screens”. For the purposes of this book we will treat the terms as the same and focus specifically on creating an application that works cross-platform.

1.2 *Deciding between Native and Cross-Platform*

Within various development circles stands a big debate, go native or go cross-platform. While the final answer is always unique to the team and developers there are some major points that need to be taken into consideration before making a final decision. I do want to point out that either way you can create some great applications.

For native development some positive reasons to use native code are the execution speed, ease of access to core or custom features, and final package size. However the downside to going with native code is the limited reuse of code, longer development cycles for projects requiring multiple platforms, and more languages that your teams must be proficient in to successfully execute an application.

With cross-platform development some positive reasons to use a cross-platform language are the development speed, consistency of application across devices, time and cost savings,

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=796>

and finally only having to master or use a single language. As with any decision we do have some downsides. For cross-platform development this usually means that it is harder to access core platform features, the final package size is usually larger to support multiple platforms, and the code's execution time is typically slower as there is a level of abstraction between the device and our code (see figure 1.1).

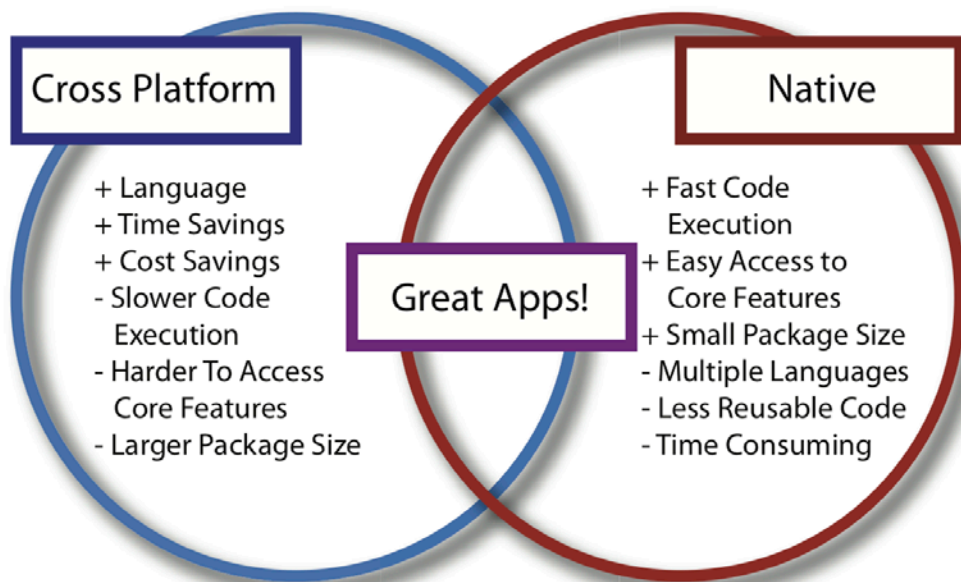


Figure 1.1 – Cross-Platform vs Native Development

It is the points of time and cost savings that I want to focus on. As a consultant, all of my clients want to create an amazing application while minimizing cost and maximizing their return on investment. As such many of my clients start the application development process by focusing on a singular device platform that they want to target, usually Android or iOS. If you are creating an application for a singular device you have the option to either use native development techniques or cross-platform development techniques.

At some point though my clients always eventually ask the question, “can my application developed for platform X also work on platform Y?” Immediately the only answer is to use cross-platform development without needlessly ballooning the development cost and the amount of time necessary to fully support multiple platforms.

As you’re reading this book I am assuming you’re interested in cross-platform development, specifically Flex Mobile.

1.3 What is Flex Mobile?

With the decision to use a cross-platform framework behind us for our mobile applications we will look at Flex Mobile specifically.

Some time back, before Flex 4.5+, Flex was already the best user interface framework on the market for browser and desktop applications. Mobile devices were starting to take off and Adobe was planning to create a second lightweight framework influenced by Flex for mobile devices, codenamed Slider. During the development and exploration of this lightweight framework the Flex SDK team found the best possible solution, combining the lessons learned from the lightweight framework and optimizing the full Flex framework.

Providing the Flex framework we know and love with the mobile optimizations of a mobile specific framework, along with some industrial upgrades to Flash Builder, the Flash Platform teams released Flex 4.5. Including components specifically for mobile, Flex 4.5 took existing mobile components appropriate for the mobile user experience and provided mobile skins size appropriate for touch input and multi-dpi layouts. The codename for this merged mobile-capable framework was “Hero”.

Powered by hundreds of active developers and contributors, the Flex framework is always improving and expanding. At the time of this writing Flex version 4.6 is just being released, adding additional mobile components to the Flex framework. The following figure shows just some of the Flex components skinned for mobile use (see figure 1.2).

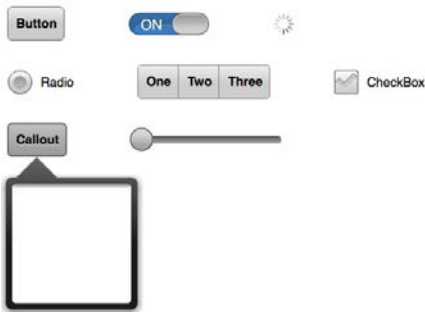


Figure 1.2 – Small Sample of Flex Visual Components Skinned for Mobile Interaction

Along with the selected list of mobile components shown there are many other components that are harder to visualize. While this definition is simple enough it opens the door to many more questions. What language am I coding in? What runs our code? Next we will be answering these questions.

1.3.1 What language am I coding in?

When developing a Flex application you will hear people use phrases like “the MXML code”, or “the ActionScript code”, or even “the Flex code”. Even poor helpless recruiters will send out job requests insisting the developers have 3 years of “MXML” development.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=796>

So what is right?

Flex is a framework built using the ActionScript language. Whether you are writing out MXML markup or ActionScript code you're always writing in ActionScript. You'll notice that I used the words "MXML markup" rather than "MXML code". This is because MXML is an XML based markup protocol created by Adobe for the Flex framework to make the layout of visual components easier to read. As its popularity grew other ActionScript frameworks used the concept of MXML markup to use a tag-based markup to describe their ActionScript objects.

It doesn't look like ActionScript to me.

MXML is purely an XML representation of the ActionScript objects. All of the rules of XML still apply such as namespaces and markup formatting. When compiling our application the compilers within the Flex framework take our MXML tags and convert them to ActionScript classes. After this translation is complete then the compiler creates our file to be run on our runtime.

Can I see the generated code?

It's a little too advanced to go into right now but if you want to be able to see the generated code created you can always add the `-keep-generated-actionscript=true` in your compiler settings.

http://livedocs.adobe.com/flex/3/html/help.html?content=compilers_14.html

With a better understanding for the language we need to answer the question of what executes our code?

1.3.2 What runs our code?

A term you've probably heard before is *runtime environment*, also sometimes referred to as just a *runtime*. A runtime environment is a collection of code, settings, and program that executes the code we write. When developing with Flex there are two runtimes to be aware of: the Flash Player Runtime and the Adobe Integrated Runtime (AIR).

FLASH PLAYER RUNTIME

The Flash Player runtime is the program that runs to execute our compiled application within an Internet browser. The Flash Player is available for every major Internet browser and provides a consistent runtime and code executing experience. As wonderful as the Flash Player can be, for this book we won't be concerning ourselves with the Flash Player. The applications we will create all with run on the AIR runtime.

AIR

The AIR runtime was created to provide a consistent runtime for installed applications running outside of the browser. Gaining access to system resources previously not available from the browser these applications can be installed on mobile devices, TVs, and desktops. All the applications we create in this book will run on the AIR runtime for testing and deployment. The growing list of mobile devices that support Adobe AIR include Android, iOS,

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=796>

and QNX (BlackBerry) devices. While it may not be obvious when I list Android devices I am also including the Nook, Kindle Fire, and other various Android tablets and phones.

Within this book we will utilize these components and many others as we develop a handful applications learning the finer points of mobile using the Flex framework. The best way to understand Flex and the power behind it is to jump in headfirst.

1.4 Hello World

In chapter 2 we will immediately get into application development, before we do it is always nice to create a quick Hello World example to set expectations for our development environment and show how quickly we can display some text with the Flex framework. Hello World examples are always helpful mini-applications that show how quickly we can go from nothing to being able to display the words “Hello World” to a user. This is the first Flex Mobile application that we will be creating.

1.4.1 Prerequisites

For this book you don’t have to be a Flex expert, you only need to have an interest in making great mobile applications that can run across multiple platforms. However there are some basic expectations we will be using throughout this book.

YOUR IDE

While many of the examples within this book use Flash Builder for our Integrated Development Environment (IDE) you don’t have to. An IDE is the application you use to code your application. The “I” in IDE states that the coding application isn’t created for just one language, but many different languages all within the same environment. If you’re used to developing with FDT, TextMate, IntelliJ, or any other of the Flex and Flash capable IDE feel free to use it. It is outside of the scope of this book to setup or customize these IDEs however full code examples will be shown throughout this book that are 100% IDE agnostic.

FLEX VERSION

The only real requirement of this book is that you are developing with at least Flex version 4.6. While many of the examples and topics within this book will work with Flex 4.5, there are a few code examples in the later half of the book that cover some of the newer capabilities of Flex that require Flex 4.6. All versions of Flex beyond 4.6 will work with the examples shown in this book.

MOBILE DEVICES

If you are one of the lucky developers to own multiple mobile devices then you can enjoy using each one of your devices throughout this book. If you are missing a QNX (BlackBerry), iOS, and/or Android device don’t worry, Flash Builder – or your favorite IDE – can provide you a simulator to run your code. Not all device capabilities are supported within the device simulators but you can still develop using these simulators. Obviously as you get closer to a final release you’ll want to pick up and test your application on each device you intend to release on.

1.4.2 Creating a new Application

In chapter 2 we will go step by step in the application creation process, looking at every option and selection – including how to run our application within the simulator. For this part we will quickly run through how to create a mobile application using Flash Builder.

Using Flash Builder the first step is to select to create a new “Flex Mobile Project” (see figure 1.3).

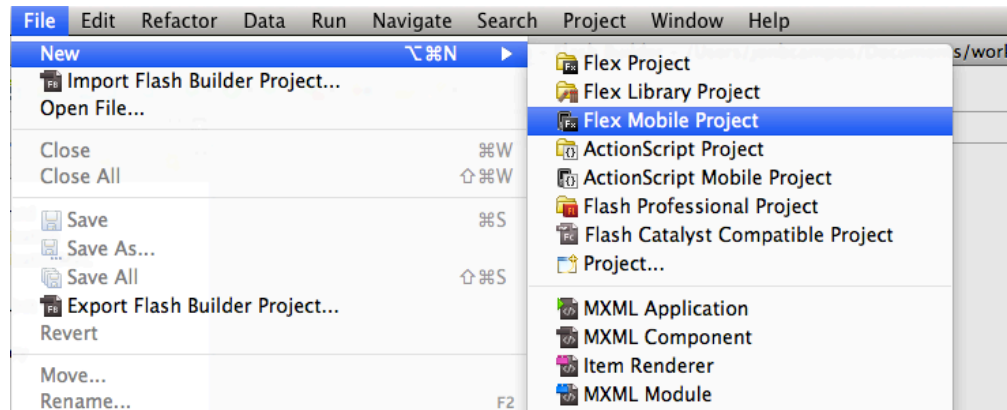


Figure 1.3 – New Flex Mobile Project

After selecting to create a new “Flex Mobile Project”, Flash Builder will lead us through the final steps to start up our new project (see figure 1.4). The first step in creating a new project is always to name the project and determine where we want the project to be saved.

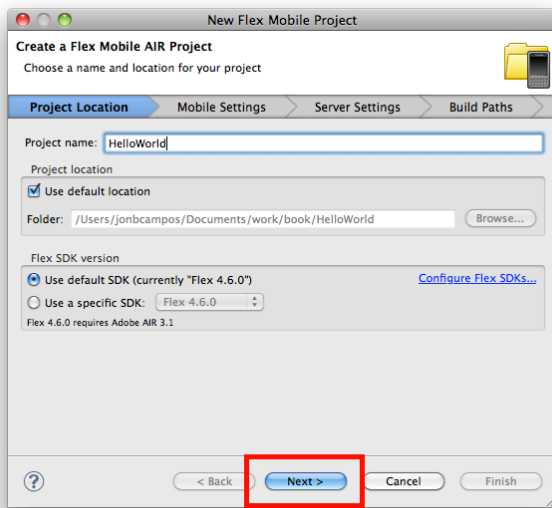


Figure 1.4 – Project Location Dialog Box

After naming our new Flex Mobile Project “HelloWorld”, select “Next >”, to configure our application (see figure 1.5).

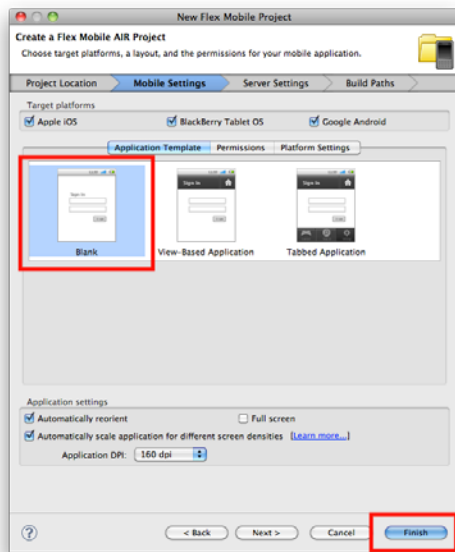


Figure 1.5 – Configure Application Dialog

In step two we can skip most of the project settings – we will revisit these settings in chapter 2 – and select to create a “Blank Application” Template. With this one setting change we just need to hit finish and Flash Builder will create our application template.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  applicationDPI="160">
  <fx:Declarations>
  <!-- Place non-visual elements (e.g., services, value objects) here -->
  </fx:Declarations>
</s:Application>
```

Our last change to add in the Hello World text is to include a Label component, displaying text onscreen.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  applicationDPI="160">
  <fx:Declarations>
  <!-- Place non-visual elements (e.g., services, value objects) here -->
  </fx:Declarations>
  <s:Label text="Hello World"/> #a
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=796>

```
</s:Application>
```

```
#a "Hello World" Label
```

That's it. We've completed our Hello World application. As simple as this application is we can already see how quickly we can go from no code to an application that we "could" run on multiple platforms. Obviously we aren't going to release this application for other people to download but our example showed us just how comfortable this process would be. If you want to know more in chapter 2 we will expand on our Flex knowledge and really customize our application.

1.5 Introducing Our Running Example

Working with a public API can give your applications the important data necessary to populate an application. An API, or Application Programming Interface, is simply a defined process detailing how to interact with data available from another software system. For the purposes of learning mobile application development it is important to be able to make service calls, show lists of data and details, along with integrating device specific capabilities into our application to utilize the abilities provided to us by the device.

The main application created within this book is a Rotten Tomatoes application displaying movie data provided by the Rotten Tomatoes API. The Rotten Tomatoes API will provide us the ability to quickly get access to a huge library of movie data and we will be able to extend the mobile experience by adding new mobile specific capabilities to our application. In this section I will introduce what a public API is, how to integrate with an API, explain what data Rotten Tomatoes will provide to us, and finally lay out a roadmap for our mobile application.

1.5.1 The Roadmap for our Example

For this book we will create an application using the Rotten Tomatoes API, with help from the RottenTomatoesAS3.swc, that will run on phones and tablets on the Android, QNX and iOS platform.

Starting simple our application will pull and display data from the Rotten Tomatoes API. As our application matures we will update our application to run across screens of various DPI followed by giving the application enterprise strength overhaul utilizing the popular Robotlegs MVC framework.

We'll round out our application by giving it access to some extra native capabilities with Native Extensions just after giving us the ability to make money using integrated ads.

Ending everything our application will include a full unit test suite and a series of build scripts to quickly create release quality files to upload to the various app markets.

There are plenty of stops along the way and additional features to be had; this is just a mile-high roadmap of where we are going.

1.5.2 Public APIs and Rotten Tomatoes

While trying to reach to a larger audience many companies open up their data to other application, including ours, using a specified protocol called an Application Programming

Interface. Using protocols like XML and JSON our application can quickly integrate with these other applications and their data, expanding the features available within our application.

For our application we will be utilizing the Rotten Tomatoes API provided by Flixster.

WELCOME TO THE ROTTEN TOMATOES API

The API gives access to Rotten Tomatoes' wealth of movie information, allowing anyone to build applications and widgets enriched with Rotten Tomatoes data.

Using the API, users can, for example:

- Search for movies and retrieve detailed movie information, like cast, directors, and movie posters
- Access to the Rotten Tomatoes Score (aggregation of critic's scores) and the Audience Score
- Get the current box office movies, new releases, and upcoming movies

From: <http://developer.rottentomatoes.com/>

To fully utilize the Rotten Tomatoes API you will need to sign up for an API Key with Flixster. Signing up for an API Key is free and only takes a few minutes.

SIGN UP FOR AN API KEY

To sign up for an API Key go to: <http://developer.rottentomatoes.com/>

From this page you must first register for an account, then register your application for an API Key. Once Flixster accepts your application you'll have your API Key (see figure 1.6).

Get Started Right Now!

Follow the steps below to start using the Rotten Tomatoes API:

- Register for a user account. ←
- Apply for an API key. ←
- Browse the [documentation](#) (or try our [dynamic documentation!](#)).
- Join a discussion in the [forums](#).

Figure 1.6 – Register and Signup for an API Key

We won't have to start from scratch though. Provided for you we will be using the RottenTomatoesAS3 API to quickly integrate the Rotten Tomatoes data without having to make complicated service calls and JSON to ActionScript translations.

1.5.3 The RottenTomatoesAS3 API

Available from github, the RottenTomatoesAS3 API provides an extremely easy way to pull data from Rotten Tomatoes. The full source is available for viewing and forking at:

<https://github.com/jonbcampos/RottenTomatoesAS3>

In chapter 2 we will discuss what a SWC is and how to use it to consume code. If you already know or just want to pull a required resource use the following link to quickly get the compiled SWC at:

<https://github.com/downloads/jonbcampos/RottenTomatoesAS3/RottenTomatoesAS3.swc>

What is a SWC?

We will look closer into what a SWC is later, how to create one, and the features provided by a SWC but for now just know that a SWC is a file that contains some code and other assets packaged up to be easily shared between projects.

To see a live example of the RottenTomatoesAS3 API in action please check out Tour De Flex (<http://www.adobe.com/devnet/flex/tourdeflex.html>). Tour De Flex is a wonderful application containing examples for many different available APIs, components, and code samples (see figure 1.7).

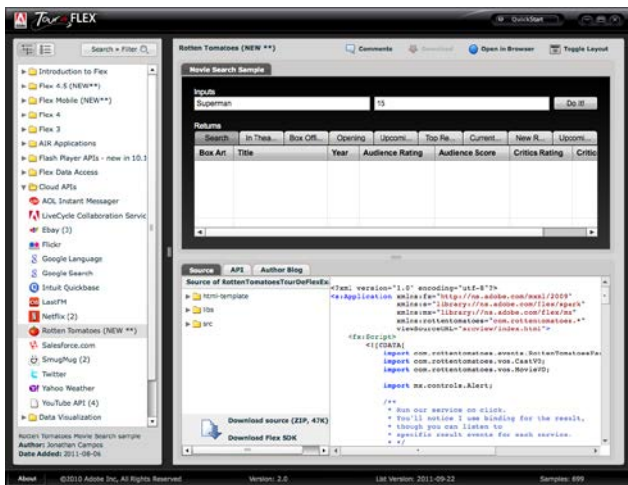


Figure 1.7 – Tour De Flex Application with the Rotten Tomatoes API selected

Understanding how to get our hands on the data from the Rotten Tomatoes API and where to get the ActionScript code making our lives easier we can move on and look at what we will be creating throughout this book.

1.6 Summary

We have a long way to go and the journey is just starting. In this chapter we looked at some key terms in the growing mobile debate: native development vs cross platform development. From there we looked at how Flex, originally developed as a component library to build rich internet and desktop application, grew to include many new components created to support mobile development on a variety of platforms. Then we lightly looked at how to start up a new Flex Mobile Project with Flash Builder and introduced where we are going throughout the rest of the book. The most important take away from this chapter should be a high-level understanding of the Rotten Tomatoes API and getting our own API Key with Flixster. With these steps complete we just need to pull the RottenTomatoesAS3 SWC file and move onto chapter 2 where we will get right into some real development.

Key takeaways:

- Understand the terms Multi-Screen, Multi-Device, and Cross-Platform
- Quickly make a Hello World Example
- Learn about the Rotten Tomatoes API and our Rotten Tomatoes Application