How To Create An App

Tom Seymour, Minot State University, USA
Jasmine Zakir Hussain, Minot State University, USA
Sharon Reynolds, Minot State University, USA

ABSTRACT

Mobile Computing is booming and everyone has ideas how to profit from it. But only few people know how to build an app that will make those ideas a reality. Businesses are looking for innovative ways to generate revenue from this rapidly growing technology while taking the advantage of the immediacy and convenience that mobile technology has to offer. Mobile apps play a vital role, and have changed the focus from what's on the Web, to the apps on mobile devices, and it is no longer an option but an imperative. The purpose of this article is to provide information on how to create an app and to explore the commonly used tools and technologies to create an app.

Keywords: Mobile Apps; Mobile Computing; Mobile Device; Mobile Technology

INTRODUCTION

Today, we are in a technological world in which almost everything is done using a mobile device from banking, entertainment, education, in which app developers build software apps and users consume the apps in an environment provided by the app store. Mobile apps are a new revenue-generating stream. Within the app world, developers may evolve their strategies, adapting to the requirements of users and producing new apps that fit into ever-changing niches. Forrester defines mobile applications as "Software on a mobile device such as a smartphone, tablet, eReader, or any other device that users can easily take with them and use anywhere, anytime." Smartphones combine the functions performed by conventional handheld PC, or personal digital assistants (PDAs), with the ability to make phone calls. These devices permit the user to make phone calls, pick up and send email, manage calendars, keep a to-do-list and address book up to date, take pictures, and be entertained with games, music, and video, as well as a variety of other functions with new applications being created on a regular basis.

Mobile applications are big business from Apple's App Store to Android's play store. Windows and Blackberry market hundreds of thousands of apps that are created and downloaded every second. Developers are increasing challenges to develop apps that generate revenue. Downloadable software apps have sparked a growth surge in the software industry: Apple introduced the iPad to U.S. buyers in April 2010, and within a few days after its launch, more than 3000 new applications were available for downloading, in addition to the 150,000 apps originally developed for the iPhone. One reason for this rapid growth is there are virtually no barriers to entry. Another is that in October 2009, iPhone developers were told they could give away their applications on an experimental basis and ask for payment later. By late 2009, Yahoo didn't have an App Store, but it listed apps for downloading on its home page (Boehret, 2010; MacMilan et al., 2009).

Apple's App Store had an exceptional year, with the technology company announcing its customers spent more than $10 billion on mobile applications in 2013. In fact, December 2013 alone saw almost 3 billion app downloads with customers spending more than $1 billion making it the most successful month in App Store history. These numbers are seen as a positive sign, despite a recent Gartner report predicting by 2018, less than 0.01 percent of consumer-facing mobile applications will be considered a financial success by their developers (Website Magazine, 2014, February). A new survey of 302 marketing and digital development executives, conducted by Forbes Insights, and sponsored by Adobe showed marketing executives see the value in extending mobile channels not only to customers, but also to employees and partners. The survey finds 53% of marketing executives designing and deploying mobile apps, both internal- and external-facing, as part of their marketing and business development strategies. Most are building and releasing apps not only for customer consumption, but also to better connect employees and partners.

The mobile browser is evolving from a thin rendering engine to a sophisticated application delivery platform running complex JavaScript applications. HTML5 will be the best option for a widely available, platform-neutral application delivery technology that is able to deliver sophisticated applications with a good-quality user experience. However, issues such as performance, fragmentation, and immaturity are still a challenge for developers to develop quality apps. Three platforms iOS, Android, and Windows are gaining significant market share in the smartphone, tablet, and PC space. "Although more than 100 'platform independent' development tools exist, most involve technical or commercial compromises, such as lock-in to relatively niche technologies and small vendors. This will drive increasing interest in HTML5 as a somewhat-standardized, widely available, platform-neutral delivery technology," says Gartner's Ken Dulaney, vice president and distinguished analyst. Further, Gartner predicts that through 2014, improved JavaScript performance will begin to push HTML5 and the browser as a mainstream enterprise application development environment. Gartner recommends that developers focus on creating expanded user interface models including richer voice and video that can connect people in new and different ways.

Google's App Inventor is one of the tools designed to allow people without programming experience to create Android apps and games for phones and tablets. GameSalad is a similar application for Mac OS, which streamlines the process of creating Web, iPhone, and iPad games where non-programmers have found success in the notoriously crowded App Store. What App Inventor and GameSalad do is automate the creation of code, hiding it from view.

TYPES OF APP

The proliferation of mobile devices and platforms represents a game-changing technology shift on a number of levels. Organizations must decide not only the best strategic use of mobile platforms, but also how to most efficiently implement them. Basically there are three different application approaches: native, web, and hybrid.

Native App

Native apps are installed through an application store such as Google Play or Apple's App Store. Native apps live on the device and are accessed through icons on the device home screen. They are developed specifically for one platform, and can take full advantage of all the device-specific hardware capabilities such as the accelerometer, gyroscope, compass, camera, microphone, or even third-party accessories through the 30-pin dock connector or Bluetooth. Native apps are specific to the device programming language as Objective-C for iOS, Java for Android, and C# for Windows, etc. Native applications can also use frameworks to access data from on-board applications such as Contacts, Calendar, Mail, or Maps and are usually developed using an integrated development environment (IDE). IDEs provide tools for building, debugging, version control, and other professional development tools. In a nutshell, native apps provide the best usability, the best features, and overall the best mobile experience.

Example: Angry Birds, DinnerSpinner

Mobile Web App

Web apps are not real applications as they are really websites that, in many ways, look and feel like native applications, but are not implemented as such. They are run by a browser and use standard web technologies—typically HTML5, CSS, and JavaScript. Users' first access them as they would access any web page. Then they navigate to a special URL and have the option of installing apps on their home screen by creating a bookmark to that page. Mobile Web applications differ from native applications as they use web technologies and are not limited to the underlying platform for deployment. An important part of the "write-once-run-anywhere" HTML5 methodology is that distribution and support is much easier than for native apps. For a native app, there are longer development and testing cycles, after which the consumer typically must log into a store and download a new version to get the latest fix. While developers can create sophisticated apps with HTML5 and JavaScript alone, some vital limitations remain at the time of this writing, specifically session management, secure offline storage, and access to native device functionality.

Example: Financial Times Web App, Allreceipes.com

Hybrid App

Hybrid apps are part native apps and part web apps. Both apps use both native phone and web features and are considered hybrid apps. Like native apps, they live in an app store and can take advantage of the many device features available. Like web apps, they rely on HTML being rendered in a browser; with the caveat that the browser is embedded within the app. Hybrid apps are also popular because they allow cross-platform development such as the same HTML code components can be reused on different mobile operating systems, reducing significantly the development costs. Tools such as PhoneGap and SenchaTouch allow people to design and code across platforms, using the power of HTML. Gartner predicts that by 2016 more than 50 percent of mobile apps will be hybrid, as they are platform-neutral.

Example: Twitter, Yelp

TECHNOLOGY

Native App

Objective C for iOS

Objective-C is the primary programming language used when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime. Compared to other object-oriented languages based on C, Objective-C is very dynamic. Objective-C supports an open style of dynamic binding, a style that can accommodate a simple architecture for interactive user interfaces.

Java for Android

Android applications are developed using the Java language. As of now, that's the only option to develop native applications for Android. Java is a very popular programming language developed by Sun Microsystems (now owned by Oracle). Developed long after C and C++, Java incorporates many of the powerful features of those powerful languages while addressing some of their drawbacks. Still, programming languages are only as powerful as their libraries. These libraries exist to help developers build applications. Android relies heavily on these Java fundamentals. The Android SDK includes many standard Java libraries like data structure libraries, math libraries, graphics libraries, networking libraries as well as special Android libraries that will help you develop awesome Android applications.

Web App

Hypertext Markup Language (HTML5)

HTML is the way that website developers mark up text, images, and other types of content to tell Web browsers what to do. HTML5 is the latest standard for HTML. HTML5 was designed to replace HTML 4, XHTML, and the HTML DOM Level 2. It was specially designed to deliver rich content without the need for additional plugins. The current version delivers everything from animation to graphics, music to movies, and can also be used to build complicated web applications. HTML5 is also cross-platform. It is designed to work whether you are using a PC, or a Tablet, a Smartphone, or a Smart TV.

Cascading Style Sheets (CSS)

CSS stands for Cascading Style Sheets. This addition to HTML gives developers and users more control over how to display pages. Using CSS, you can create one style sheet that (for example) sets rules for how a header should look, and then you can apply those rules to dozens or even thousands of pages without having to type in everything by hand. CSS is the language for telling Web browsers how to format HTML documents. You can use CSS to set all kinds of webpage characteristics, such as font styles and sizes for the different types of text, the alignment of different elements in the document, and the page's background color.

JavaScript

JavaScript is a scripting language that can be used to create more interactive and dynamic Web content. The purpose of JavaScript is to make the app more dynamic by giving it the ability to manipulate data and to interact with the user. That's a type of programming language that controls other application such as a Web browser. JavaScript code runs in your Web browser rather than on a server and we call it a client-side scripting language. JavaScript is rapidly getting faster—so fast, in fact, that HP Palm WebOS 2.0 rewrote its services layer from Java to the extremely popular node.js platform (http://nodejs.org/), which is built on Google's V8 engine to obtain better performance at a lower CPU cost (and therefore longer battery life). The trend we're seeing is the Web technology stack running at a low level, and it's in production today on millions of devices.

CREATING YOUR FIRST APP

Design

The starting point for any new app is identifying the problem the app is going to solve. In this step visualize the main features and the approximate layout and structure of the application. Visualization of the app may help the individual or the team to understand the mission and give control of the entire execution process. Wireframing is the process of creating a prototype of the app. There are a number of prototyping tools available online such as Balsamiq, Moqups, and HotGloo which allows you to not only drag and drop all your placeholders and representative graphics into place, but also add button functionality so that you can click through your app in review mode. Designing a good user interface is the next step to creating a successful app. Apple introduced a new method for designing user interfaces in Xcode 4.2 and iOS 5 called Storyboard. Storyboard lets you design and implement your interface in a single step using a graphical environment. You can see exactly what you're building while you're building it, get immediate feedback about what's working and what's not, and make instant visible changes to your interface. After you've created a user interface, you define how users can interact with what they see by writing code to respond to user actions in your interface.

Implementation

After the user interface is designed, the next step is to make the interface work by adding the backend logic. iOS apps are event-driven programming which means the flow of the app is determined by events which may be a system event or user actions. Basically, the user performs the actions on the interface which triggers events in the app. Most of the event handling logic can be defined in view controllers. The implementation of the logic is programmed using the Objective C programming language. After you implement your app's behavior, you create a data model to support your app's interface. An app's data model defines the way you maintain data in your app. Data models can range from a basic dictionary of objects to complex databases. A good data model is essential to creating a solid foundation for your app. As you start implementing your model, remember you don't have to implement everything from scratch. There are a number of frameworks that provide existing functionality for you to build upon. For instance, the Foundation framework includes classes representing basic data types—such as strings and numbers—as well as collection classes for storing other objects. It's recommended that, where possible, you use existing framework classes—or subclass them to add your own app's features—instead of trying to reimplement their functionality.

Testing

User interface testing is the most critical part of the app development process. One must create an ad hoc provisioning certificate to test the app on the device. An ad hoc provisioning profile is a distribution provisioning profile for iOS apps that allows your app to be installed on designated devices and use key technologies and services without the assistance of Xcode. It's one of the two types of distribution provisioning profiles you can create for iOS apps. An ad hoc provisioning profile ensures that test versions of your app aren't copied and distributed without your knowledge. When you're ready to distribute your app to testers, you create an ad hoc provisioning profile specifying an App ID that matches one or more of your apps, a set of test devices, and a single distribution certificate. Each iOS device in an ad hoc provisioning profile is identified by its unique device ID (UDID). The devices you register and add to a provisioning profile are stored by Member Center. Each individual or company can register up to 100 devices per membership year for development and testing. You sign the iOS App Store Package containing your app using the distribution certificate specified in the ad hoc provisioning profile and distribute iOS App Store Package to testers.

The Android framework includes an integrated testing framework that helps you test all aspects of your application and the SDK tools include tools for setting up and running test applications. Whether you are working in Eclipse with ADT or working from the command line, the SDK tools help you set up and run your tests within an emulator or the device you are targeting. The SDK tools for building and tests are available in Eclipse with ADT, and also in command-line form for use with other IDEs. These tools get information from the project of the application under test and use this information to automatically create the build files, manifest file, and directory structure for the test package. Android test suites are based on JUnit.

Publishing

Signing up for a Developer Account

To publish the app in the App Store or the Play Store, a developer account is a primary requirement. To develop an iOS app, create an account with iOS Development Center. It costs $99 for a year and requires you to provide the tax and bank account information of your business or yourself. Create a distribution certificate in order to distribute your iOS application; the account holder is responsible for creating a distribution certificate. This certificate is distinct from a development certificate and only this certificate will enable enterprise app distribution. The next step in creating a provisional profile is creating Distribution Provisioning Profiles which match your distribution certificate, and allows you to create applications that your users can run on their device. You create a provisioning profile for a specific application, or multiple applications by specifying the AppID that is authorized by the profile. If a user has an application, but doesn't have a profile that authorizes its use, the user isn't able to use the application. Because these profiles are tied to your certificate, when you revoke your certificate or it expires the application will no longer run on the device.

To distribute an application through Google Play, a developer account must be created. For an Android developer account, visit Google developer console and register for an account. There is a one time $25 registration fee charged for a Google Play Developer Console account. All applications need to be signed with a cryptographic key that expires after October 22, 2033. The maximum size for an APK published on Google Play is 50MB. If an application exceeds that size, Google Play will allow extra assets to be delivered through APK Expansion Files. Android Expansion files permit the APK to have 2 additional files, each of them up to 2GB in size. Google Play will host and distribute these files at no cost. Promotional assets have to be prepared like launcher icons, high resolution application icons, screen shots of the app, promotional graphics, and feature graphic. Upload the APK (Android Application package) and complete the listing details. There are various publishing options available. For example, you will be able to choose what country the app may be distributed. Complete the consent details as this is the mandatory section and is used to declare the application meets the Android Consent Guidelines and acknowledgement that the application is subject to US export laws.

NATIVE iOS DEVELOPMENT

Most native iOS applications are written in the Objective-C programming language, and developers typically use Xcode to develop their applications. To build a native iOS app using Mac OS X is important, other operating systems are not supported. The development tools that you'll need are iOS 7 SDK and Xcode 5. You can run the app you build in the iOS simulator, which is part of the iOS SDK.

Developer Tools

Xcode

Xcode is an IDE (Integrated Development Environment) used by iOS developers to build applications. Designed from the ground up to take advantage of the newest Apple technologies, Xcode integrates all the tools you need to develop quality iOS apps. The unified interface smoothly transitions from composing source code, to debugging, and even to designing your next stunning user interface, all within the same window. The Xcode workspace keeps you focused while you work. Enable features such as iCloud or Game Center and Xcode automatically generates the provisioning profile for you. When ready, Xcode will security sign your app and directly submit it to the App Store. It's not just a code editor but it has a variety of additional capabilities which include: auto complete support, static code analysis - it finds bugs in your code before you compile, memory leaks, and a variety of debugging and performance tools. Xcode automatically configures your apps to use the latest Apple services, manages your many images in a unified asset catalog, and helps you design an app that looks gorgeous on iOS 7 or OS X Mavericks. You can download the latest version of Xcode for free from the App Store on your Mac. (The App Store app is installed with OS X version 10.7 and later. If you have an earlier version of OS X, you need to upgrade.) The iOS SDK is included with Xcode.

Once you have installed Xcode, the next step would be to create a new project and enter some basic information about the app such as product name, organization name, company identifier, and class prefix. The value you enter in the class prefix tells Xcode to attach a unique prefix to every class that you generate with Xcode. Since Objective C does not support namespacing as in java attaching, a unique prefix to the class will avoid naming conflicts. The “Devices” setting lets you restrict your application to run only on an iPhone or an iPad and selecting the “universal” option will enable the application to run on both.

Navigation and View Controllers

The screen functionality of iOS applications is grouped into what are known as view controllers. A view controller contains the logic needed to interact with the controls on a screen. It also interacts with another component called the navigation controller, which in turn provides the mechanism for moving between view controllers. Apps that contain structured content can use navigation controllers to navigate between levels of content. The navigation controller itself manages the display of one or more custom view controllers, each of which manages the data at a specific level in your data hierarchy. The navigation controller also provides controls for determining the current location in this data hierarchy and for navigating back up the hierarchy. There are other controllers available; Apple documentation gives detailed information about the entire available controllers.

Storyboard

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers. Xcode provides a visual editor for storyboards, where you can lay out and design the user interface of your application by adding views such as buttons, table views, and text views onto scenes.

Apple LLVM Compiler

Apple's next generation compiler technology, the Apple LLVM compiler, does more than compile code. Apple LLVM technology is integrated into the entire development experience. The same parser used to build C/C++

and Objective-C code powers Xcode’s indexing engine, providing relevant code completions and even related documentation. As you work, Apple LLVM is constantly evaluating what you type, identifying coding mistakes shown as Live Issues, and thinking ahead for ways to Fix-it for you. Other compilers can tell you what is wrong — Apple LLVM can make it right.

Interface Builder

Interface Builder is an application that lets you build your interfaces visually. Built-in objects like buttons, tab bars, sliders, and labels can easily be dragged onto your app's interface and then configured by tweaking the palettes and panels. You can also use Interface Builder to connect targets and actions as to what happens when an interface element is acted on by a user, and what object handles the action as well as manipulate controllers and object bindings.

iOS Simulator

The iOS Simulator runs your application in much the same way as an actual iOS device. Because it is quick to launch and debug, the iOS Simulator makes for a perfect test bed to make sure your user interface works the way you intend, your network calls are correct, and that the interface properly rotates with an orientation change. You can even simulate touch gestures by using the mouse. The iOS Simulator is a great time saver.

Coding

Even for the most basic of applications to function, some code must be written. Storyboard, helps to create a user interface and the interactions between the view controllers but the actual logic to make the interface work is written using the Objective C programming language. That is all done by you, the developer, in Objective-C. When an application is running, the application delegate handles its overall lifecycle. Various methods in this delegate are called when key events in the application’s lifecycle occur. These events could be any of the following such as the start of the application, the application is moved to the background, the application is brought to the foreground, and the application is about to be terminated, and a push notification arrives. The next area of attention is the view controller. Just as with the application delegate, the view controller has its own lifecycle. The main code for the application is in the ViewController.m file.

Design the User Interface

You create your app’s user interface in Interface Builder. Select a user interface file in the project navigator, and the file’s contents open in Interface Builder in the editor area of the workspace window. A user interface file has the filename extension .storyboard or .xib. Default user interface files are supplied by Xcode when you create new projects from its built-in templates. To add user interface elements, drag objects from the utility area into Interface Builder, where you arrange the elements, set their attributes, and establish connections between them and the code in your source files. As you lay out your app’s user interface elements in Interface Builder, you can write the code that implements their behavior in the assistant editor.

Connect User Interface Objects to Code

To access the storyboard objects from the code, you must define the relationships between them. You write the code that implements the behavior of your user interface objects. Your code communicates with user interface objects through action and outlet connections. Create an action connection when you need to send a message from a control to your code. A control is a user interface object that causes instant actions or visible results when the user manipulates the object. When the user clicks a button, for example, the button sends an action message telling your code to execute an appropriate action. Text fields, sliders, and switches are examples of commonly used controls in iOS; checkboxes, scroll bars, and text fields are commonly used controls in Mac OS. Create an outlet connection when you need to send a message from your code to a user interface object. The object can be a control or any other object defined in your storyboard of xib file, such as a label, progress indicator, or map view. When your code determines that a button should disappear, for example, your code sends a message through an outlet telling the button to hide itself.

Send Action Messages from a Control to Your Code

Whenever the user activates a control, such as by tapping it, the control should send a message instructing your code to perform some action. The easiest way to configure a control to send an action message to your code is by Control-dragging from the control in Interface Builder to your object’s implementation file.

Send Messages to a User Interface Object Through an Outlet

To enable your code to send messages to a user interface object (telling a label to display a text string, for example, or telling a button to appear or disappear), connect the user interface object to an outlet in your code. The easiest way to make an outlet connection is by Control-dragging from a user interface object in Interface Builder to the implementation file for another object.

Building the App

To build and run your iOS or Mac app, choose a scheme and a run destination in the workspace toolbar, and click the Run button. Clicking the Stop button causes your app to quit. If you are running an iOS app, Xcode launches it either in iOS Simulator or on an iOS device connected to your Mac. If you are running a Mac app, Xcode launches it directly on your Mac.

Choose a Scheme to Build Your App

A scheme is a collection of settings that specify the targets to build for a project, the build configuration to use, and the executable environment to use when the product is launched. When you open an existing project (or create a new one), Xcode automatically creates a scheme for each target.

Choose a Destination to Run Your App

When you build an app, the destination determines where the app runs after it’s built. For Mac apps, the destination is the Mac on which the app is built. For iOS apps, the destination can be a provisioned iOS device connected to the Mac, or iOS Simulator. Installed as part of the Xcode tools along with the iOS SDK, iOS Simulator runs on your Mac and simulates an iPhone or iPad environment.

Run the App

Click the Run button in the workspace toolbar to compile, link, and execute your code. If the app builds successfully, Xcode runs it and starts a debugging session. Depending on your destination, Xcode launches your iOS app either in iOS Simulator or on a connected iOS device. If there are errors during the compilation or link phase, Xcode doesn’t run your code.

Testing the App

Create unit tests that automatically exercise the features of your application. Monitor the results of the tests and fix any issues from the test navigator. You can use the Xcode service, available in OS X Server, to automate the execution of unit tests. From Xcode on your development Mac, you create bots that run on a separate server. In addition to running unit tests, bots automatically perform static analysis on your code, build your app, and archive it for distribution to testers or the App Store. Bots help you ensure that your product is always in a releasable state—and when there's a failure, the service notifies you or the person whose code change caused the failure.

Unit tests are programs that automatically exercise the features of your application and check the results. Unit tests are often used to detect regressions introduced by code changes to a project. Some developers write unit tests first, and then implement methods that pass the tests. A unit of code is the smallest testable component of your project—for example, a method in a class or a set of methods that accomplish an essential purpose. A test case exercises a unit of code in a specific way; if the result of the test is different from the expected result, the test case fails. A test suite is made up of a set of test cases. You create test suites based on the XCTest framework. You can develop one or more test suites to test different aspects of your code.

Use Snapshots to Restore Project Wide Changes

Snapshots are archives that include the current state of all project and workspace settings and all document files in the project. Snapshots provide an easy way to back up the current version of your project or workspace. If something goes wrong because of a code change you make, a snapshot makes it easy to restore your entire project, even a deleted project, to a previous state. Xcode automatically creates a project or workspace snapshot before you perform any mass-editing operation, such as refactoring your code or executing a Find and Replace operation. You can also set Xcode to automatically create snapshots in other circumstances, such as before a build, by choosing Xcode > Preferences, selecting Behaviors, and selecting the Create Snapshot option. You can also create a snapshot manually by choosing File > Create Snapshot.

NATIVE ANDROID APP DEVELOPMENT

Developer Tools

Android SDK

The Android SDK provides the API libraries and developer tools necessary to build, test, and debug apps for Android. Android Development Tool (ADT) bundle includes the essential Android SDK components and a version of the eclipse IDE with built in Android development tools to streamline the Android apps. A new Android development environment called Android Studio, based on IntelliJ IDEA, is now available as an early access preview.

Setting up the ADT Bundle

Download the latest version of the ADT bundle at http://developer.android.com/sdk/index.html. Install the SDK and Eclipse IDE. Unpack the ZIP file (named adt-bundle-<os_platform>.zip) and save it to an appropriate location, such as a Development directory in your home directory. Open the adt-bundle-<os_platform>/eclipse/ directory and launch eclipse.

Coding

Define the User Interface

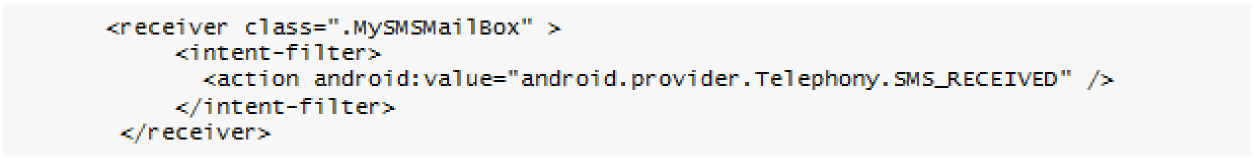
To start with coding the first step must be to define the user interface. The UI for an application is generally defined as a series of layout files. These are XML-based files that describe the controls on a screen and the relationship of their layouts relative to one another. These are placed in the project in a directory structure defined by the Android SDK. At runtime, Android dynamically loads content from this directory structure. In order to get the interface working Java code must be written to respond to various events that occur from the controls on a given screen and from changes in the lifecycle of an application. Java code is also responsible for loading the layout and menu files associated with each screen. The Android SDK was designed from the outset to support a wide variety of device capabilities. Much has been written about the fragmentation of devices on the Android platform. The device attributes include screen size, pixel density, and Android API versions. Screen layouts, image assets (called drawables), styles and other configuration files are all incorporated in a series of subdirectories underneath a master “resource” directory. In the process of defining the layouts, you should associate an ID value with each control (or UI widget) that you wish to reference from code. This can be done by specifying an ID value in the properties inspector in the layout editor. In iOS, we loaded the logic specific to a given screen into a ViewController. In Android, these separate screens are treated as separate “activities.” Just like in an iOS UIViewController, there is a defined life cycle for an activity that governs when the activity starts, pauses, resumes, and stops. It’s up to the developer to place code in the various methods of the activity to respond to the various states of the lifecycle.

Activity

The activity is the most visible and prominent form of an Android application. An activity presents the user interface to an application with the assistance of a class known as a view. The view class is implemented as various UI elements, such as text boxes, labels, buttons, and other UIs typical in mobile computing platforms. An application may contain one or more activities. They are typically on a one-to-one relationship with the screens found in an application. An application moves from one activity to another by calling a method known as startActivity() or startSubActivity(). The former method is used when the application desires to simply "switch" to the new activity. The latter is used when a synchronous call/response paradigm is desired. In both cases, intent is passed as an argument to the method.

Intent Filters

The core components of an application (its activities, services, and broadcast receivers) are activated by intents. An intent is a bundle of information (an Intent object) describing a desired action — including the data to be acted upon, the category of component that should perform the action, and other pertinent instructions. Android locates an appropriate component to respond to the intent, launches a new instance of the component if one is needed, and passes it to the Intent object. Components advertise their capabilities — the kinds of intents they can respond to — through intent filters. Since the Android system must learn which intents a component can handle before it launches the component, intent filters are specified in the manifest as <intent-filter> elements. A component may have any number of filters, each one describing a different capability. The following snippet comes from an Android application using intent filter that responds to incoming SMS (text) messages:



```
<receiver class=".MySMSMailBox" >
     <intent-filter>
       <action android:value="android.provider.Telephony.SMS_RECEIVED" />
     </intent-filter>
 </receiver>
```

Figure 1: A Code Snippet of IntentFilter

ContentProvider

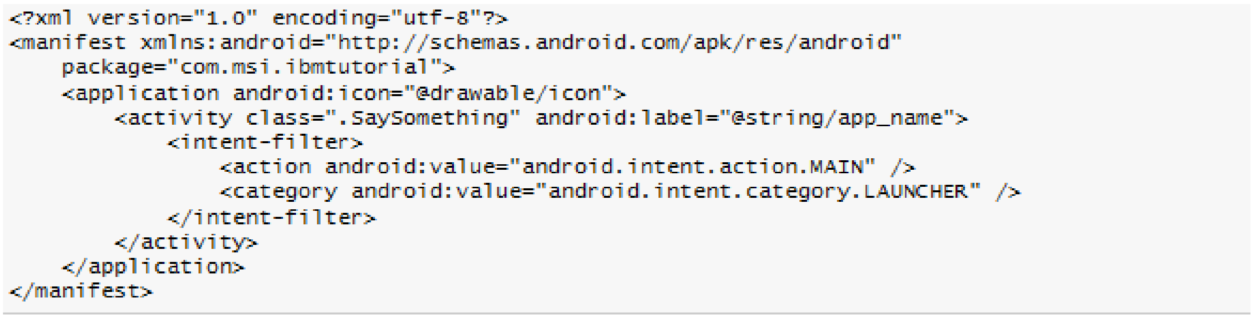
The ContentProvider is the Android mechanism for data-store abstraction. For instance, the address book contains all the contacts and phone numbers a person might require when using a mobile phone. The ContentProvider is a mechanism to abstract access to a particular data store. In many ways, the ContentProvider acts in the role of a database server. Operations to read and write content to a particular data store should be passed through an appropriate ContentProvider, rather than accessing a file or database directly.

Services

Like other multitasked computing environments, there are applications running "in the background" that perform various duties. Android calls these types of applications "services." The service is an Android application that has no User Interface.

AndroidManifest.xml

Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.msi.ibmtutorial">
    <application android:icon="@drawable/icon">
        <activity class=".SaySomething" android:label="@string/app_name">
            <intent-filter>
                <action android:value="android.intent.action.MAIN" />
                <category android:value="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figure 2: Sample AndroidManifest.xml file

The manifest does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities (for example, which Intent messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the Instrumentation classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is being developed and tested; they're removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

Testing

Android applications are tested using the Android emulator or on a regular device such as a phone or tablet. The emulator enables you to emulate a variety of Android versions, as well as different screen aspect ratios and resolutions. While testing your application on a real device is always a good idea, the emulator helps you to test on device configurations that are not readily available. To use the emulator, you must create an Android Virtual Device. Think of this as a virtual phone or tablet. Creating multiple AVDs, each with a unique configuration, is possible to test your app on different screen sizes and memory capacities. To test the application on your own device, you'll need to enable USB debugging on the device.

MOBILE APPLICATION FRAMEWORK

Adobe's PhoneGap

Adobe's PhoneGap platform enables a developer to create an app that runs on a variety of mobile devices. The developer accomplishes this largely by writing the user interface portion of their application with Web technologies such as HTML, CSS, and JavaScript. PhoneGap's development tools then bundle the HTML, CSS, and JavaScript files into platform-specific deployment packages. PhoneGap supports a wide variety of platforms as iOS, Android, Windows 8, Blackberry, WebOS, Symbian, and Tizen. PhoneGap essentially wraps a Web view of your HTML, CSS, and JavaScript in a native application. This is required because the Web view in an application does not inherently support many device features, such as access to the file system or the camera. PhoneGap has a bridging mechanism that allows JavaScript running in the Web view to invoke native code contained in the application.

PhoneGap or Cordova: What's the Difference

Cordova is the open-source project that is the basis of PhoneGap. Apache Cordova is a set of device APIs that allow a mobile app developer to access native device function such as the camera or accelerometer from JavaScript. Combined with a UI framework such as jQuery Mobile or Dojo Mobile or Sencha Touch, this allows a smartphone app to be developed with just HTML, CSS, and JavaScript. In most cases, the names may be used interchangeably. With "PhoneGap" already being trademarked, a new name was needed when the project was open-sourced — hence, Cordova. "PhoneGap" continues to be used by Adobe for commercial products.

Using PhoneGap

The majority of PhoneGap's capabilities lie in non-visual components — things that access the file system, network availability, geolocation, etc. Applications written for mobile browsers must respect the limitations of the given mobile platform (processing speed, screen size, network speed, touch events, etc.). Unless you have been working with HTML and CSS for a long time and are well aware of these issues, developing an effective mobile application without some sort of framework can be daunting. Here are just some of the offerings in this area some popular ones are JQuery Mobile, Ionic, Sencha Touch, Kendo UI Complete, AppGyver's Steroids, Enyo, and TopCoat. One distinction to be made is that some frameworks support a wider variety of devices and device versions than others. Some mobile frameworks are built on a particular MVC platform. For example, Ionic is built on the AngularJS framework. This might make a particular library more attractive to developers who are already familiar with the respective MVC framework. Frameworks such as Sencha Touch abstract the DOM from the developer through the use of APIs, freeing the developer from having to worry about details of browser vendor implementations. Sencha also provides graphic development tools, such as Sencha Architect, to aid the development process. Sencha provides commercial support and training, too, which are worth investigating.

Installing PhoneGap

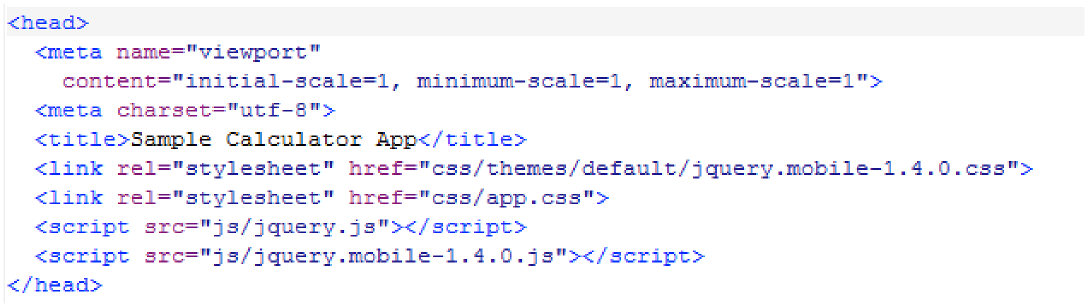
With the introduction of PhoneGap version 3, installing PhoneGap's tools and setting up projects have been greatly simplified, using the new command-line interface (CLI). The command-line tools are based on node.js; you must install node.js before installing PhoneGap. Once node.js is in place, simply run one command for the node package manager to install PhoneGap's tools:

```
npm install -g cordova
```

The command-line package for Cordova provides only the PhoneGap portion of the tools that are necessary for development. You will still need the mobile SDKs for the platforms that you wish to support.

Coding

For creating the app, I use JetBrain, which is an excellent WebStorm IDE for Web development because it is well suited for PhoneGap. Unlike with native platforms, there is no vendor-supplied integrated development environment (IDE) for PhoneGap that is similar to Xcode or Eclipse. The user interface is shaped by index.html. This one file contains the entire HTML used by the application.



```
<head>
  <meta name="viewport"
    content="initial-scale=1, minimum-scale=1, maximum-scale=1">
  <meta charset="utf-8">
  <title>Sample Calculator App</title>
  <link rel="stylesheet" href="css/themes/default/jquery.mobile-1.4.0.css">
  <link rel="stylesheet" href="css/app.css">
  <script src="js/jquery.js"></script>
  <script src="js/jquery.mobile-1.4.0.js"></script>
</head>
```

Figure 3: A Code Snippet of index.html using JQuery Mobile

The viewport metatag scales the width of the page to match the width of the device's browser. If this is not present, then the mobile browser wouldn't know the page is optimized for mobile use and would zoom out to display the page as though it were on a desktop browser. Next, load two style sheets. The vast majority of the CSS is contained in the jquery.mobile.css file, with a few minor application-specific CSS overrides in the app.css file. The head section then loads the scripts needed to support jQuery and jQuery Mobile. Unlike the native apps, in which they are separate view controllers or activities for each screen, here we'll place the HTML markup for both screens in one file. JQuery Mobile supports the concept of a page being divided into sections that make it appear to the user as though they are navigating between screens.

JQuery Mobile takes care of performing the transitions between these virtual pages in the application. Because much of the user interface and interaction between the controls is defined declaratively in our HTML, we have considerably less code to write to get the application running. For more complex apps, jQuery Mobile also enables you to dynamically load new pages, rather than place the entire markup in one file. This is the preferred approach if your application has several screens, because maintaining the entire markup in one file would become difficult and the mobile browser would have to parse that large document before rendering anything. Forcing the browser to process a large document would slow down the startup time of your application.

Building the App in the Cloud using PhoneGap Build

To build an application for a particular platform, you must install the SDK for that platform on your machine. This could be a problem if, for instance, you're on a Mac and want to target Windows tablets, whose SDK requires you to be on a Windows machine. Adobe offers a service named PhoneGap Build to help in this situation. PhoneGap Build enables you to upload a ZIP file containing the HTML, CSS, and JavaScript of your application. Additionally, PhoneGap 3 enables you to submit an application to PhoneGap Build right from the command line; from there, PhoneGap Build takes over and produces a deployment bundle for the desired platform.

Once the build process is complete, you may either download the deployment bundle and manually install it to your device or take a picture of a QR code on PhoneGap Build's website and download the deployment package directly to your mobile device. What's more, PhoneGap Build supports a feature named Hydration, which enables the application to download updates to the HTML, CSS, and JavaScript files whenever it launches. This means that testers will always be running the latest version of your application, without having to go through the traditional updating process. Note that, at least for now, Hydration is meant for development and testing — not for final production code that will be submitted to the app store.

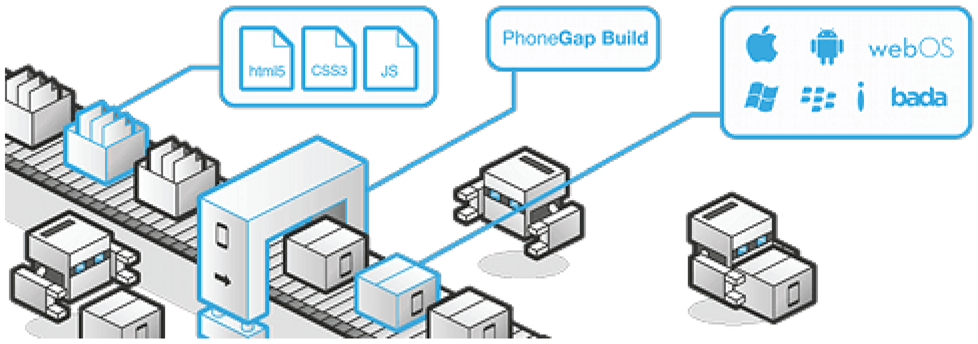


Figure 4: Adobe PhoneGap build

APPCLERATOR TITANIUM

Appcelerator lets developers create rich native apps for every major mobile platform, all from a single JavaScript code base. And with live prototyping and fully integrated automated testing, designers, developers, and testers can collaborate to deliver great apps at mobile speed. Appcelerator Studio is an open, extensible development environment for creating beautiful native apps across different mobile devices and Operating Systems including iOS, Android, and BlackBerry, as well as hybrid and HTML5.

Appcelerator Studio IDE

Appcelerator Studio IDE – a powerful Eclipse-based IDE, based on Titanium Studio IDE, simplifies the development process, allowing enterprise developers to rapidly build, test, package, and publish mobiles applications across multiple devices and Operating Systems to both public and enterprise app stores. Appcelerator Studio IDE also includes unique enterprise capabilities such as a profiler, LiveView, and a code analyzer. With integrations to common source control and build systems, developers can also streamline their development activities by working in a single environment. Appcelerator Studio IDE integrates with Appcelerator Test to provide automated regression testing and with the most common public and enterprise app stores to speed the release process.

Appcelerator SDK

Appcelerator SDK – an open SDK, based on the Titanium SDK, with over 5,000 device and mobile operating system APIs. Enterprise developers can create native, hybrid, and mobile web apps through a JavaScript-based SDK using a single code base. Appcelerator Alloy is an open MVC development framework, for building high quality mobile applications. It follows a model-view-controller (MVC) architecture and using XML and CSS provides a simple model for separating the application user interface, business logic and data models. With Appcelerator Alloy you can employ a component-based development process allowing the creation of standalone packages of pre-built functionality (both client and server side).

WEB VIEWS – WHAT’S THE DOWNSIDE?

Not all Web Views Render the Same

The capability of Web views has improved markedly in the last two years. However, your user base may not be running the latest and greatest version of Android or iOS. The Android platform has generally lagged behind the iOS platform in the capability and performance of its Web view. Google appears to be putting in a renewed effort to improve that situation, but users stuck on old versions of Android (especially releases before Ice Cream Sandwich 4.x) might find that the performance of Web views, especially those with complex layouts, do not match what is achievable on iOS. Developers will need to test the HTML5 code on Android 2.2, 2.3, and 4.x to ensure that content renders consistently. The major downside is in creating a common look and feel between the two worlds. For example, a native list view (default white on black, with block elevator bar) looks very different to a web list view unless you put lots of effort in replicating the native behavior in your CSS.

Performance

Web views are typically not as performant as native code. However, in many situations, you may find the performance to be perfectly acceptable, particularly on current hardware. Keep in mind that bridging calls between the native and Web view portions of your code will exact a performance toll. You won’t want to make calls across this bridge at sub-second intervals, but for most applications this is not an issue.

Security

Web should be treated with some suspicion, particularly if it will be interacting with an interface to your native code. Ensure you’ve exposed only those portions of your native code that are truly required for interaction with the Web layer. Make sure you take standard Web security practices into account to ensure your Web content cannot be compromised.

CONCLUSION

The future of mobile applications development seems to be filled with opportunities as many organizations have already realized the benefits of becoming mobile enterprises. Having a mobile app along with the mobile-friendly website is found to be more beneficial. Consumers are spending more time in operating mobile apps rather than interacting with browsers on a PC. Obviously there are many factors that comprise an enterprise's mobility strategy. These factors include programmers' skills, device functionality, and the importance of security and interoperability while hybrid apps are an important element enabling enterprises to advance their strategies. Web-based applications may never replace native applications; they will retain an edge in business applications that need rapid development. It's clear that in the fast-moving mobile world, it's better to start with what is available now rather than to wait too long and fall behind to a point where you can't catch up.

AUTHOR INFORMATION

Dr. Tom Seymour earned his PhD degree at Colorado State University, his Masters of Arts degree at the University of North Dakota, and his Bachelor of Science degree at Mayville State University. He has given over 150 Computer/E-Commerce presentations in 41 states and 10 foreign countries. Tom teaches technology classes in the classroom and via the Internet. Dr. Seymour has won the following awards: Distinguished Alumni Award - Mayville State University (2009), Legislative Award - Information Technology Council of ND (2008), Peer Reviewer Award - Higher Learning Commission (2007), a Legislative Award - NDAEYC - Children's Champion Award (2005) and Distinguished Professor Award - Minot Chamber of Commerce (1989) -The IACIS Ben Bauman Award of Excellence (2011). E-mail: seymour@minotstateu.edu (Corresponding author)

Jasmine Zakir Hussain is a graduate student obtaining her Master's Degree in Information Systems at Minot State University in Minot, North Dakota. She has a Bachelor's Degree in Electrical and Electronics Engineering, Anna University, India. She has experience in design and implementation of Java/J2EE based applications and has been working for Verizon Communication. Jasmine also has knowledge of mobile applications development, which is of Android skill base. Jasmine has the Above & Beyond Award (2007) from Verizon for her outstanding contribution to project implementation. E-mail: jasminezakir@outlook.com

Sharon Reynolds is an assistant professor at Minot State University and has been teaching in the Business Information Technology Department for many years. She has presented many seminars, written articles, authored and published educational books and consults with people in the community. E-mail: sharon.reynolds@minotstateu.edu

REFERENCES

1. Ableson, F. (2008). *Develop Android applications with Eclipse*. Retrieved 02/26/2014 from http://www.ibm.com/developerworks/opensource/tutorials/os-eclipse-android/
2. Adobe's PhoneGap Build. *Package mobile apps in the cloud*. Retrieved 02/27/2014 from https://build.phonegap.com
3. App Development Process. *Apple documentation*. Retrieved 02/20/2014 from https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/AppDevelopmentProcess.html#//apple_ref/doc/uid/TP40011343-CH4-SW1
4. Apple Developer Documentation. *Tools you'll love to use*. Retrieved 02/25/2014 from https://developer. apple.com/technologies/tools/
5. Apple.com. *Build breakthrough apps for your employees*. Retrieved 02/26/2014 from http://www.apple.com/business/accelerator/
6. AppManifest. *Android documentation*. Retrieved 02/23/2014 from http://developer.android.com/ guide/topics/manifest/manifest-intro.html
7. Budiu, R. (2013). *Mobile: Native apps, web apps , and hybrid apps*. Retrieved 02/25/2014 from http://www.nngroup.com/articles/mobile-native-apps/
8. iOS Developer Library. *Apple documentation*. Retrieved 02/26/2014 from https://developer.apple.com/ library/ios/navigation/

9. Nelson, J. (2012). *Mobile sites vs. apps: The coming strategy shift*. Retrieved 02/25/2014 from http://www.nngroup.com/articles/mobile-sites-vs-apps-strategy-shift/
10. Rundle, M. *Building iOS apps from scratch*. Retrieved 02/26/2014 from http://designthencode.com/scratch/
11. Smith, G. (2013). *10 excellent platforms for building mobile apps*. Retrieved 02/26/2014 from http://mashable.com/2013/12/03/build-mobile-apps/
12. Traeg, P. (2013). *Best of both worlds: Mixing HTML5 and native code*. Retrieved 02/26/2014 from http://mobile.smashingmagazine.com/2013/10/17/best-of-both-worlds-mixing-html5-native-code/
13. Xamarin Developer Center. *Publishing an application*. Retrieved 02/26/2014 from http://docs.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_application/
14. Xamarin Developer Center. *Publishing to the App Store*. Retrieved 02/26/2014 from http://docs.xamarin.com/guides/ios/deployment,_testing,_and_metrics/app_distribution_overview/publishing_to_the_app_store/