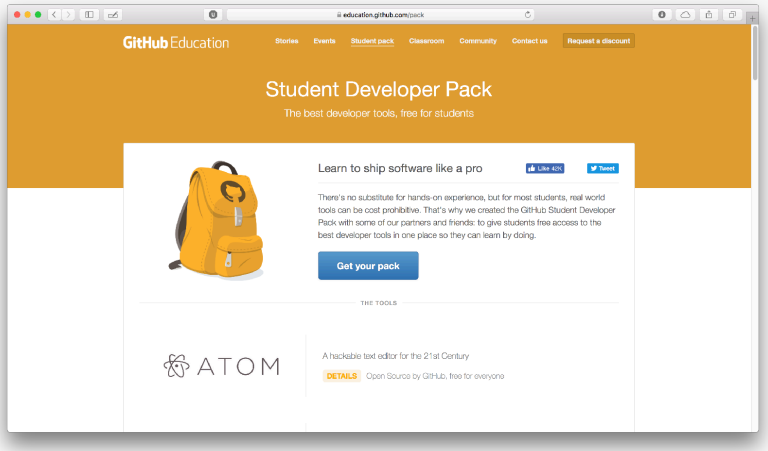
Mobile app development

Tiberiu Vilcu
Prepared for EECS 411
Sugih Jamin
2 January 2018



https://education.github.com/pack

Outline

1. Mobile app programming and MVC
2. App lifecycles
3. UI and UX
4. Working with iOS
5. Working with Android
6. Working with web apps
7. Third-party frameworks
8. Resources

Not the same as before

Developing and coding mobile apps is not like programming you've done for other classes.

It is largely event-driven and does rely only on serial code execution.

It's an opportunity to integrate and apply concepts from other CSE courses taken.

Event-driven programming

1. Define a UI with interactive elements
2. Add event handlers to UI elements
3. Add handlers for internal state changes
4. Process and store data locally and externally, use APIs
5. Test extensively

MVC

Software architecture pattern to split application in 3 parts.

Model: contains app behavior, data, logic, and rules

View: contains any output of information (visual or audio)

Controller: contains inputs elements and connects the model to the view through interaction

MVC example

Xcode and Android Studio Interface Builders are good examples.

Model: XML code that represents your UI

View: graphic representation of your UI in the editor

Controller: draggable and modifiable elements and properties

Abstract app lifecycle

Not running: not launched or terminated by system

Inactive: in foreground, running code, not receiving events

Active: in foreground, running code, receiving events

Background: not in foreground, running code, before suspension

Suspended: not in foreground, not running code, still in memory

iOS app lifecycle

- applicationDidBecomeActive(): app is about to become the foreground app
- applicationWillResignActive(): app is no longer the foreground app
- applicationDidEnterBackground(): app is running in the background
- applicationWillEnterForeground(): app is moving from background to foreground
- applicationWillTerminate(): app is being terminated

iOS UIViewController lifecycle

- viewDidLoad(): called when controller is created
- viewDidLayoutSubviews(): called whenever a subview needs to layout
- viewWillAppear(): called when view is about to show, used to customize view settings
- viewDidAppear(): called after the view shows
- viewWillDisappear(): called before the view is removed
- viewDidDisappear(): called after the view is in a disappeared state

Android app activity lifecycle

- onCreate(): activity first loads up
- onStart(): activity becomes visible to the user
- onResume(): activity comes to foreground and user interacts
- onPause(): user is leaving activity (but not destroying)
- onStop(): activity is no longer visible to user
- onDestroy(): activity being destroyed by user or system

User interface and user experience

To a user, UI/UX is the most important first impression.

Their perception is strongly influenced by your UI.

Think of your code in terms of actions taken by the user (and the OS)!

Tips when building UIs

- Start simple: minimal views, minimal elements, minimal colors
- Plan your views/pages ahead of time
- Use a UI builder before doing it in the actual app (later lecture)
- Don’t hide important content or interactive elements
- Follow Apple’s Human Interface Guidelines or Android User Interface Guidelines

Things to know for iOS

- You can only develop in Xcode on macOS
- Apple Developer Account costs $99 (yearly)
- You should work primarily/only in Swift
- You can and should simulate on all various screen sizes
- Certain features are only on newer version of iOS if you need them
- TestFlight can be used for beta testing (even A/B)

Tips before you begin on iOS

- You can learn Swift as you go
- You will find bugs in Xcode; save/commit often and google problems
- Xcode can automatically convert old Swift code into new code (however it's not guaranteed to work or be correct)
- Plan out data caching and storage ahead of time
- Don’t waste too much time on an app icon

Development resources

CocoaPods: dependency manager for Swift and Objective-C; automatically creates Xcode workspace

Carthage: dependency manager for binary frameworks; does not automatically modify project/build settings

Third-party SDKs: use of most public APIs require custom SDKs; these can be downloaded and build via the dependency managers

Apple Developer account

You will eventually need an account for the class.

- $99 for a one-year subscription
- Required to publish to App Store
- Gives access to betas
- Gives ability to run/save your app on actual phones

Consider splitting accounts with other teams to save costs.

Publishing an iOS app

1. Create Apple Developer account
2. Login to iTunes Connect and create an “app record”
3. Create an archive in Xcode and run validation tests
4. Upload in Xcode to iTunes Connect
5. Choose a build, add app information, and submit for review on iTunes Connect

Be wary of content restrictions on the app store.

Reviews usually take a few days.

Warnings about iOS / Swift

- Things change quickly in Swift! (currently 4.0)
- A lot of code online is outdated
- SDK’s online also often outdated or use Objective-C
  You may need to build them yourselves
- Your entire team should be on the same versions of Xcode/macOS/iOS

Things to know for Android

- You can develop in Android Studio on any OS
- Google Play Developer Console costs $25 (one-time)
- You will primarily use Java and XML
- You should test on different phone emulators

Tips before you begin on Android

- Learn how to use Java and XML as you go
- Plan your activities and how they interact ahead of time
- Name your widgets and variables carefully; refactoring can cause problems
- Don’t waste too much time on icons

Publishing an Android app

1. Sign up for a Developer Console with your Google account
2. Build and sign a release version in Android Studio
3. Create an application on the Console and update the store listing, content rating, and distribution
4. Upload your APK and publish

Warnings about Android

- Android compatibility is very fragile
- Different API versions have widely different use percentages
- Code may sometimes be underlined in Android Studio while correct as it takes time to refactor
- Build often to view errors

Mobile “web” apps

Mobile apps written in a modern web framework.

Can be written using:

- HTML
- CSS
- JavaScript (React especially)

Doesn’t require knowing gritty details of web-dev or mobile app dev.

React Native

Open-sourced software to build mobile apps from (mostly) JavaScript

- Works in JavaScript and allows UI components to be added
- Allows use of native (Swift/Java) code when necessary
- Can reload a preview of the app without recompiling
- Compiles to iOS and Android applications through Xcode and Android Studio

React Native pros

- Quicker development of two applications in parallel
- Hot reloading can save a lot of time vs. compiling
- JavaScript may be more familiar
- Easily supports A/B testing
- Supported by big companies

React Native cons

- Inconsistencies between UI elements natively supported
- Not as polished or documented as original IDEs
- Some unsupported features are hard to add in both versions
- Can cause major slowdowns in development
- Publishing apps is not the same

Apache Cordova

Open-source software to build unified web and mobile apps.

- Works with HTML, CSS, and JavaScript
- Wraps app into native container app used on mobile devices
- Supports native OS features in unified JavaScript API

Ionic

Open-source software to build unified web and mobile apps.

Built on top of Apache Cordova.

- Supports native OS features in unified JavaScript API
- Includes front-end components that makes it look more like native app
- More performance optimized
- Pro features for enhanced development

Ionic pros

- Quicker development of two+ applications in parallel
- JavaScript, HTML, and CSS for those familiar
- (More) native plugins that interact with OS
- Live reloading, A/B testing, and custom style/icons
- Large online community

Ionic cons

- There are limitations to features supported on the OS
- Does not match the native look of iOS/Android apps
- Still creates hybrid, not fully native, apps
- Can cause major slowdowns in development
- Publishing apps is not the same

Picking an OS

1. Do a majority of teammates have access to a Mac?
2. Is there an OS-specific feature to take advantage of?
3. Is it a gaming app built through another platform?
4. Is there a demographics difference between users?

Picking a platform

1. Is there a specific hardware feature not supported by web apps?
2. Will there be a complicated UI with many custom points of interaction?
3. Will third-party SDKs be used?
4. Will front-end lag be an issue?
5. Is there experience with Swift/Java?

React Native vs Ionic

- Builds a native app in Swift/Java
- In theory is “faster”
- Has a more native OS-specific look, important for users
- Is used by big companies
- Defines UI by HTML/CSS, not JS
- Has licensed style and icons
- Has good API documentation and community resources

Previous lectures

- iOS sample project repo: https://github.com/UM-EECS-441/ios-project-sample-f17
- Android sample project repo: https://github.com/UM-EECS-441/android-project-sample-f17
- iOS development lecture slides: http://web.eecs.umich.edu/~sugih/courses/eecs441/f17/03-iOSDevelopment.pdf
- iOS development lecture recording: https://leccap.engin.umich.edu/leccap/viewer/r/AwY18V
- Android development lecture slides: http://web.eecs.umich.edu/~sugih/courses/eecs441/f17/05-AndroidDevelopment.pdf
- Android development lecture recording: https://leccap.engin.umich.edu/leccap/viewer/r/QWmqlN
- WebApp development lecture slides: http://web.eecs.umich.edu/~sugih/courses/eecs441/f17/06-WebAppDevelopment.pdf
- WebApp development lecture recording: https://leccap.engin.umich.edu/leccap/viewer/r/eMalm9

iOS Resources

- Swift guide: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html#//apple_ref/doc/uid/TP40014097-CH3-ID0
- Apple Human Interface guidelines: https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/
- App life cycle: https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html
- View controllers: https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/WorkWithViewControllers.html
- Apple developer enrollment: https://developer.apple.com/programs/enroll/
- Submitting app to App Store: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html

Android resources

- Android Studio: https://developer.android.com/studio/index.html
- Android programming tutorial: https://developer.android.com/training/index.html
- App activity lifecycle: https://developer.android.com/guide/components/activities/activity-lifecycle.html
- Google Play developer account: https://support.google.com/googleplay/android-developer/answer/6112435?hl=en
- Publishing app to Play Store: https://developer.android.com/studio/publish/index.html

Other resources

- CocoaPods: https://cocoapods.org
- Carthage: https://github.com/Carthage/Carthage
- React Native: https://facebook.github.io/react-native/
- Ionic: https://ionicframework.com
- Apache Cordova: https://cordova.apache.org