

IN4MATX 133: User Interface Software

Lecture:
Hybrid and Native
Architectures

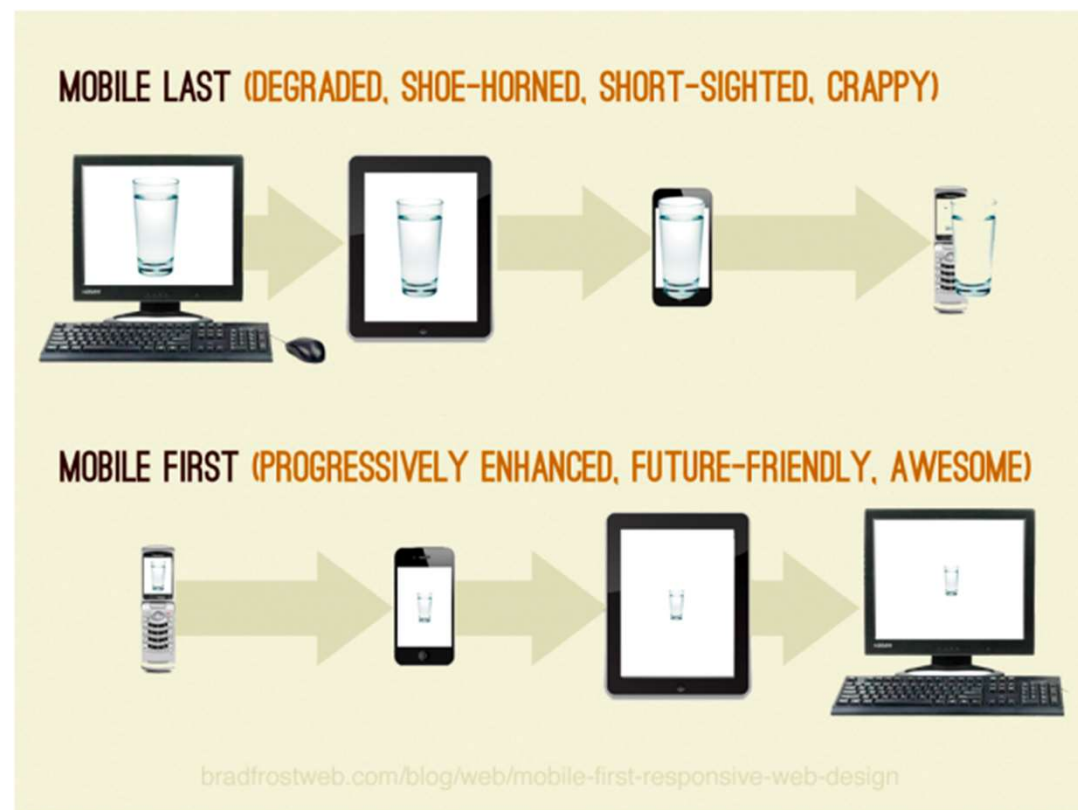
Today's goals

By the end of today, you should be able to...

- Differentiate approaches to developing mobile interfaces
- Describe advantages and disadvantages of developing native, hybrid, and web applications
- Explain which approach Ionic takes to app development

Mobile-first design

- Plan your design for mobile
- Then make your app *better* with more real estate
 - Add more features
 - Make existing features easier to navigate
- A lot of businesses make mobile-friendly websites before making dedicated apps



**Why might a business
want a mobile app
over a mobile website?**

**There are a variety of ways
to build mobile apps**

Mobile development methods

- Native
- WebView
- Hybrid
- Responsive
- Progressive Web App (PWA)
 - <https://kevinbasset.medium.com/why-havent-pwas-killed-native-apps-yet-29beca4425fa>

Native apps

- An app designed to work on a specific piece of hardware
- Usually built with tools created by the hardware or platform manufacturer
 - Android Studio for Android, in Java
 - Xcode for iOS, in Swift or Objective-C

Native apps

- As we think of them today, native apps started with the first iPhone
- Released a development platform alongside the hardware



Native apps

- iOS development languages:
 - Objective-C
 - Cocoa Touch
 - Swift
- These languages were either developed by or pretty much only used by Apple
 - Developer lock-in is a...
Disadvantage? Advantage? Both?



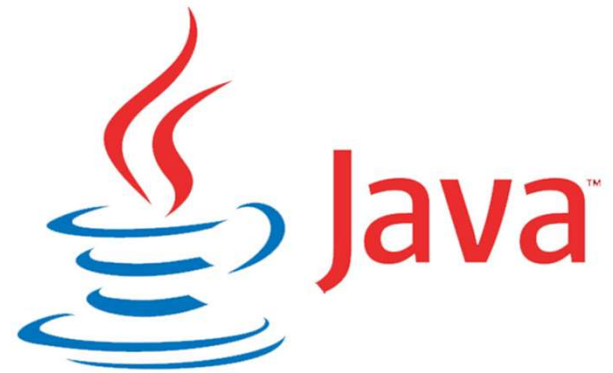
Native apps

- iOS development tools:
 - Xcode
 - iOS Source Development Kit (SDK)
 - SDK provides access to phone's storage, camera, sensors, etc.



Native apps

- Android development languages:
 - Primarily Java
 - C and C++ via Android Native Development Kit (NDK)
- Align more closely with languages used in other contexts
 - Is this an advantage? A disadvantage?



Native apps

- Android development tools:
 - Android Studio
 - Android Source Development Kit (SDK)
 - Various IDEs like Eclipse or NetBeans



Native apps

- Platform-specific codebases
 - Android is in Java, iOS is in Objective-C or Swift
 - Both use different libraries to communicate with the hardware
- Usually require starting to code from scratch



What if we already made a website for our app? Or have some other existing codebase?

What if we want to share code across phone platforms?

Solution: hybrid apps

Hybrid apps

- “Use a common code base to deploy native-like apps on a wide range of platforms”
- Two primary approaches:
 - WebView app
 - Compiled hybrid app

WebView app

- Run a webpage written in HTML/CSS/JavaScript, on the phone's internal browser
- Load that browser in a lightweight native app
- Ideally, expose some native APIs to the browser

WebView app

- Essentially, the app is just a website
- Allows the same or similar code to be used across an app and a website

WebView app frameworks

- Ionic
- jQuery mobile
- NativeScript
- These frameworks use web technologies (HTML, CSS, TypeScript, JavaScript) rather than platform-specific technologies



WebView app frameworks

- WebView apps are just websites
- What do these frameworks provide?
 - Common mobile interface elements like sliders and buttons (more on that next week)
 - The native app for running the website
 - Some APIs for communicating with platform SDKs

Compiled hybrid apps

- “Write code in one language, such as C# or JavaScript, and compile it to native code supported by each platform”
- Result: a native app for each platform
- Challenge: less freedom in development

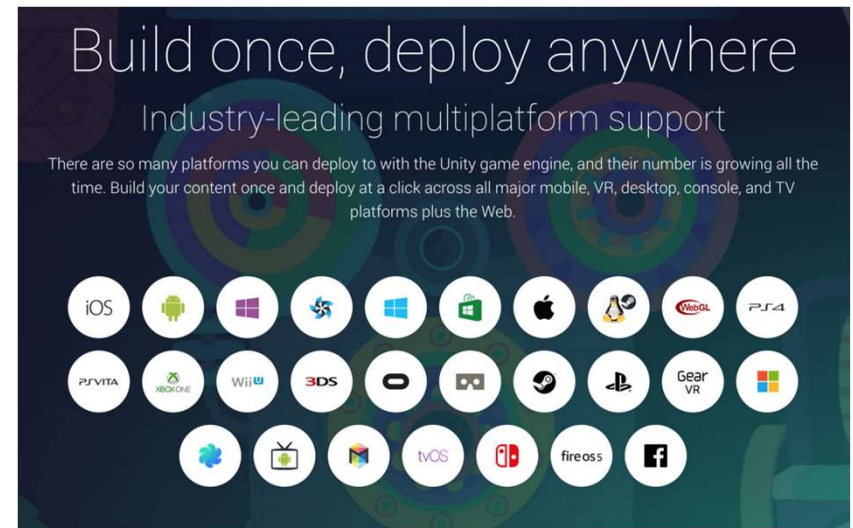
Compiled hybrid app frameworks

- Xamarin
 - C#
- Unity
 - C# of JavaScript
- React Native
 - JavaScript



Unity

- Leading game development platform
 - Supports consoles, web, and mobile
 - Will need to import or use platform-specific SDKs



React Native

- Uses React, a web framework similar to Angular
- Compiles a webpage to a native app



Performance is just one factor.
How do we choose
a development approach?

Business considerations

- Development time
- Development cost
- Maintenance concerns
- Available infrastructure

UX and design considerations

- Consistency with platform
- Device capabilities
- Interaction models supported
- Performance and usability

Technical considerations

- Programming languages
- Integration with device
- Performance
- Upkeep and maintenance
- Flexibility
- Compatibility

Pros and cons of each option

Strengths of hybrid apps

- Can share a codebase between web and mobile
- Can save time and effort (sometimes)
- Easily design for various form factors
- Access to some device capabilities

Weaknesses of hybrid apps

- Performance issues
- Inconsistency with platform
- Limited access to device capabilities

Strengths of native apps

- Consistent experience with platform
- Leverages full device capabilities
- Uses native UI elements

Weaknesses of native apps

- Need to support separate development for each platform
- Cost of app development and maintenance
- Need to learn/manage multiple programming languages
- Need to manage multiple sets of tools

Hybrid apps vs. native apps

- Hybrid apps are great when time or money is a concern and you need to deploy on multiple platforms
- Native apps are great when performance and consistency with the platform are major concerns

Hybrid apps vs. native apps

- Hybrid apps
 - News sites
 - Informational apps
 - Product showcase
 - Seasonal/one-off
- Native apps
 - Games
 - Content-heavy apps
 - Uses a lot of device resources
 - Needs specific OS capabilities

Progressive Web Apps (PWAs)

- Intended to “fill the gap” between native apps and web apps
- Really just a website that you can “install” on a phone
- Supported by major browsers & phones
- No associated framework, just a few files to add



https://en.wikipedia.org/wiki/Progressive_web_application

Progressive Web Apps (PWAs)

- Add some information to an app manifest (manifest.json)
 - Desired device orientation, URL to open, links to icons
- Relies on everything your browser relies on for other features
 - Web Storage for saving values
 - https://en.wikipedia.org/wiki/Web_storage



https://en.wikipedia.org/wiki/Progressive_web_application

Progressive Web Apps (PWAs)

- A good PWA should:
 - Start fast, stay fast
 - Work in any browser
 - Be responsive to any screen size
 - Provide a custom offline page
 - Be installable



<https://web.dev/pwa-checklist/>

Progressive Web Apps (PWAs)

- Main advantages
 - They require almost no new code or libraries, making them ideal for having a shared codebase with your website and implementing progressive enhancement
 - Most apps don't need native features
- Main disadvantage
 - They don't show up in managed app stores like Apple's App Store or Google Play, so not discoverable through traditional means
- To learn more visit (great resource for getting started):
 - https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

One Hybrid (WebView) framework: Ionic

Ionic

- WebView app framework
- Launched in 2013
- Interface implemented in Angular
 - Recently added support for React and Vue
- Capacitor is the recommended hybrid app runtime for ionic, replacing Cordova
- Apache Cordova is still supported, but not recommended for new projects



<https://ionicframework.com/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development>

Capacitor

- It provides the native app which opens the WebView
- Supports PWAs
- Also provides plugins for connecting to device resources
- Hundreds of plugins
 - Official
 - Community



<https://github.com/capacitor-community/>

<https://capacitorjs.com/docs/apis>

Ionic Native

- Ionic Native is a wrapper to bring plugins to ionic
 - Ionic Native plugins are imported as services
 - Can wrap Cordova plugins as well
- Capacitor has retro-compatibility with most Cordova plugins



<https://ionicframework.com/docs/native/>

Ionic Native

Some example plugins

- Geolocation
- Bluetooth
- Camera
- Health
- Gyroscope
- Pedometer

<https://ionicframework.com/docs/native/>

Ionic Native

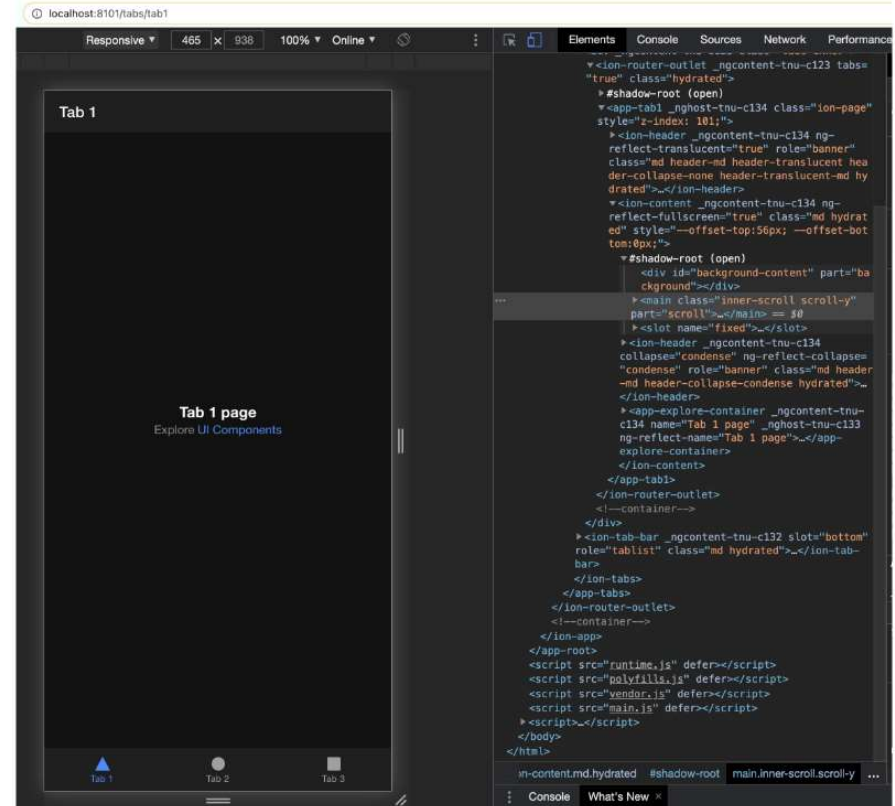
Some example plugins

- Facebook
- LinkedIn
- WeChat
- Apple Pay
- Google Maps
- Youtube

<https://ionicframework.com/docs/native/>

Ionic Dev

- Provides a WebView to open up Ionic apps
- Lets you test your Ionic app in a browser



<https://ionicframework.com/docs/cli/commands/serve>

Deploying Ionic apps

- Involves packaging up an app and “signing” it as a developer
 - For Android, this requires installing Android Studio
 - For iOS, this requires installing Xcode and getting a developer account
- Can then “deploy” the app to the app store
 - The iOS app store includes features for “beta” deployment with a small group of developers
- This process is often a pain

<https://ionicframework.com/docs/building/ios> or <https://ionicframework.com/docs/building/android>

Ionic iOS and Android Deployment

- “The key mantra of Capacitor is that developers should embrace native tools like Android Studio and Xcode”
- Pre-builds projects to be used in Xcode and Android Studio
 - Lets you test your Ionic app on an actual device or emulators
 - Emulators have limited use of plugins

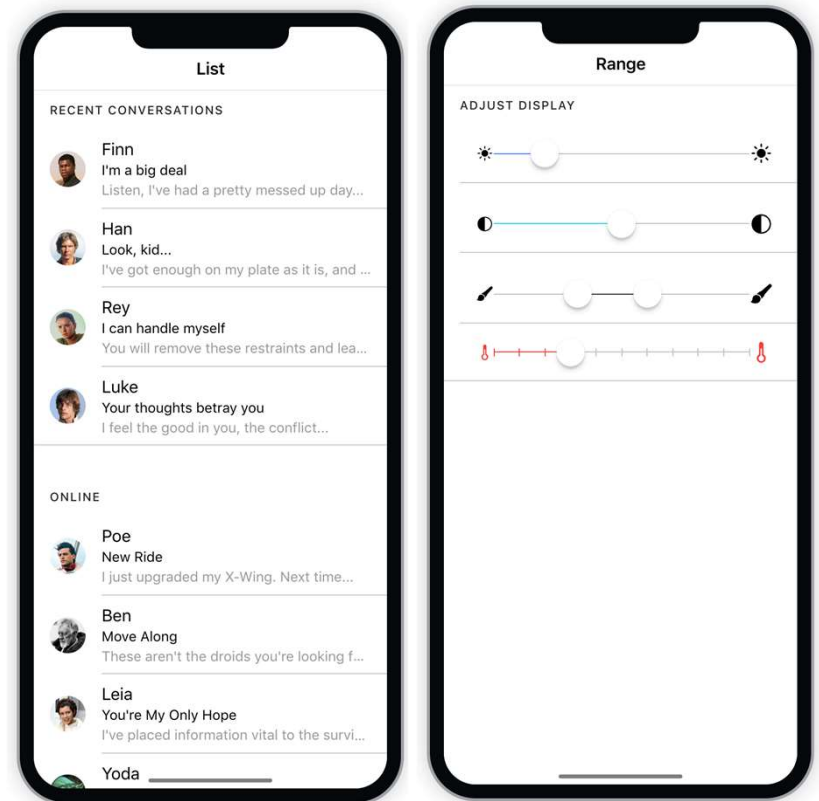
<https://ionicframework.com/docs/developing/starting>

What does Ionic add over Angular?

Ionic components

- Ionic provides Angular-style components for a lot of interface elements common in mobile interfaces
 - Lists, buttons, sliders, tabs, modal dialogs, search bars, much more
 - These are the focus of next lecture

<https://ionicframework.com/docs/components/>



Today's goals

By the end of today, you should be able to...

- Differentiate approaches to developing mobile interfaces
- Describe advantages and disadvantages of developing native, hybrid, and web applications
- Explain which approach Ionic takes to app development