CS1320
Creating Modern Web and Mobile Applications
Lecture 21
Mobile Applications I
Web Application Architecture

- Front End
- Web Browser
- HTTP
- Back End
- Web Server
- Database
- Server
- Mobile Platform
Structure of a Web Application

HTML/CSS -> Browser

Events

JavaScript

Event Handling

AJAX

URLs

Back End

HTML pages
Structure of a Mobile Application

- **Display**
- **Native Event Handler**
- **Page Handler**
- **Back End**
- **Operating System**

Events and AJAX connections.
Web and Mobile Differences

- HTML
- JavaScript Event Handler
- Browser
- URL Requests

- Front End
- Native Event Handler
- Operating System
- Page Handler
Mobile Front Ends

- **Widget-Based**
  - Hierarchy of widgets replaces HTML hierarchy
  - Text is in label widgets
  - Widgets exist for buttons, inputs, etc.
    - Corresponding to HTML form elements
  - Layout is done using layout widgets
    - These control how their contents are displayed

- **Widget Properties** control formatting and display
- **Widgets can be created and nested directly**
- **There is a language for defining widget hierarchies**
  - Generally XML-based static description
  - Differs on the different platforms
Mobile Event Handling

• The actual code is event-based
  ○ Wait for event
  ○ Act on the event by starting action that yields new events

• Events are similar to those of the browser
  ○ Based on user actions
  ○ Based on external events (timers, input ready, …)
  ○ But not quite the same (and they vary by platform)

• Event handling is written in the native language of the platform
  ○ This is what is actually executed
Browser versus Operating System

• For web applications, all interactions are with the browser
  o Mobile applications don’t use the browser, they run directly

• The functionality of the browser is replaced the operating system
  o Along with a suite of system libraries that provide functionality
  o Different platforms = Different names but the same functionality
  o Can have more functionality than the browser
    ▪ Especially for newer features of the phone
    ▪ But the browser is generally catching up
URL Requests versus Page Management

• A mobile application doesn’t go to the back end to get the next page
  o Instead it tells the operating system to switch to a different set of widgets
  o These are defined by pages (page == widget hierarchy for the page)

• Pages are akin to HTML pages
  o Can have separate code, events, etc.
  o Back on the phone goes to previous page, forward to next, …

• All pages defined as part of the mobile application
Other Mobile Differences

• Deployment
  ○ Need to go through an app store to make the app available
  ○ This can require certification and money

• Development
  ○ Development platforms for mobile apps are specific to the platform
    ▪ Android - Android Development Environment (IntelliJ extension)
    ▪ Apple - XCode
  ○ Differ from the platforms you used for mobile applications
    ▪ No browser-based debugger for example
Native Mobile Applications

• Are platform specific
  ◦ iOS for the iPhone using swift (used to be objective-c)
  ◦ Android for android phones using Java

• Use platform specific widget sets (close, but not quite matching)
• Use platform specific library calls (close, but not quite matching)
• Use platform specific environments
• How to approach native development
What You Need to Know to Write a Mobile Application

- Native language: Swift or Java
- Event set
- Platform widget set and their properties
- OS and library calls
- Page model
Default Approach: Write Multiple Front Ends

• **Pros**
  - They will look like native applications (specific to the platform)
  - They can use different UI capabilities (interactions; specific to the platform)
  - They can use different phone capabilities (latest on the platform)

• **Cons**
  - Almost all of the functionality is the same
  - Almost all of the capabilities on one platform exist on others as well
  - Want your application to be about the same on all platforms
  - More difficult to maintain multiple versions
Alternative: Write a Web (Hybrid) Application

• Front end is HTML, CSS and JavaScript
  ○ Needs to be responsive to handle different sizes
  ○ Can provide different functionality based on platform
  ○ Most of the technologies are available through HTML
    ▪ Geolocation, camera, sound, gestures, …

• A web application can be packaged to look like a native app
  ○ Screen icon, with click to start
  ○ Packaging tools exist and are easy to use

• Disadvantages
  ○ Performance is not as good as a native app
  ○ Can’t access latest OS features
  ○ Interface might not look or feel native

• Advantages
  ○ Easy to write; single platform; many apps can be done this way
Alternative: Write Once

• The target platforms are quite similar
  o Languages, APIs, capabilities
  o People/companies have realized this and made use of it

• Write the front end in language X for some X
  o Using a fixed set of libraries
  o Compile X into Java or Swift (or Objective/C); or interpret X natively
    ▪ Map library calls to library calls on native platform
    ▪ Either directly or through an intermediate library
  o Generate multiple applications from a single source
  o Still need to determine how to specify UI
    ▪ Take a common UI format and map to UI data for applications
    ▪ Take a common set of widgets and map to native widgets
Xamarin: C#

- Xamarin lets you write the app in C#
  - Using Visual Studio if desired
- Using a standard UI library (and XAML)
  - XML-specified widgets
- Using common libraries to access native APIs
- Can develop on Windows and Mac
  - Community (free) edition or Enterprise (paid)
NativeScript, React Native, Ionic: JavaScript

- Write the front end in JavaScript
  - With a static description of the user interface
- Have run as native code for
  - Apple, Google, ...
- REACT Native uses React-like constructs
- Ionic uses html, css and interfaces with
- NativeScript uses Vue (Mustache)-like constructs
Structure of a NativeScript Application

- Display
- JavaScript Event Handler
- NativeScript Libraries
- NativeScript Page Handler
- AJAX
- Back End

Events
How NativeScript Works

NativeScript Runtime

JavaScript

Application code

(N) Plugins

(N) Modules

NativeScript

Application Framework

Data binding, Navigation, Manipulation

Angular

Vue

NativeScript Core Modules

Cross-platform abstraction - Core, UI Views, Layouts

(N) Plugins

NativeScript Runtimes

Native API Access

Native

Android

iOS

(N) Plugins

Native Libraries

iOS/Native

System Frameworks

Objective-C runtime library

Plugin Native Libs and Frameworks

NativeScript CLI

• Chrome DevTools
• Safari Web Inspector

JavaScriptCore VM

Module Loader

(require) (impl.)

JS Workers

(multi-threaded JS impl.)

JS to Native Bridge

Native API metadata service

Types marshaller

Debugging

(inspector protocol impl.)

Live Sync

Function calls

Register callbacks

Call Objective-C runtime methods

Call registered callbacks

3/4/2020

19
NativeScript Basic Widgets

• Label – basic holder for strings
  ○ Also for fancy strings: FormattedString
  ○ HtmlView provides for HTML content

• Image

• Input widgets
  ○ Button, DatePicker, ListPicker, Slider, Switch, TextField, TimePicker
  ○ TextView, SearchBar, ListView, SegmentedBar

• Other
  ○ Placeholder, Progress, ActivityIndicator, WebView, ListView

• Examples: https://docs.nativescript.org/ui/overview
NativeScript Styling

• Widgets have properties that can be set explicitly or dynamically
  o But this isn’t the default way of formatting

• CSS is used to apply to widgets
  o Selectors: single widget, all widgets of a class, …
  o Properties
    ▪ All the common CSS properties are supported

• Also supports themes, less, …
Extended NativeScript Widgets

- RadSideDrawer
- RadListView
- RadCalendar
- RadChart
- RadAutoCompleteTextView
- RadDataForm
- RadGauge

https://docs.nativescript.org/ui/overview#components
NativeScript Layout Widgets

- **StackLayout** - vertical or horizontal rows
- **FlexboxLayout** - similar to CSS flex boxes
- **GridLayout** - similar to HTML tables
- **DockLayout** - around border and center
- **WrapLayout** - stack where things can wrap if needed (HTML standard)
- **AbsoluteLayout** - absolute positioning
Layout Examples

STACK  Flexbox  Grid  Dock  Wrap
NativeScript Organizational Widgets

- Page
  - Back, Forward, Load
- Dialog
  - AlertDialog, ActionDialog,
  - ConfirmDialog, LoginDialog,
  - PromptDialog
NativeScript Uses Familiar Concepts

- **Formatting**
  - Done using CSS for the widgets, rather than widget properties
  - Much of CSS is directly usable
  - Layouts match the HTML/CSS frameworks you’ve used

- **Page Contents**
  - Done using templating ala VUE
  - Context provided as part of a page description
  - Can use VUE, REACT, Angular, … as well

- **Common libraries**
  - Fetch - basically the same as fetch in the browser
Application Organization

• File Structure
  ○ Global files
  ○ Per-Page files
    ■ XML: description of the page display
    ■ CSS: CSS to format the page

• Page Rendering
  ○ Vue-like templates
  ○ User provided context (goes with the page)
Next Time

• Creating A Mobile Application Using NativeScript
Creating a Mobile Application

- Start by understanding the pages needed
- Sketch those pages
  - What they might look like
  - Interactions on the page
  - Interactions between pages
- Implement the pages one-by-one
  - Map diagram to layout widgets
  - Using sample data at first
  - Then using real data
CD FINDER

• Show using airmedia
File Organization for CD FINDER

- Pages
- Platforms
NativeScript Playground
CD FINDER Home Page

• XML
• CSS
• JavaScript
Playground And the Phone

• Connecting Playground with the phone
• Showing Changes
CD FINDER CD List Page

• XML
• CSS
• JavaScript
• Determining what the input looks like
CD FINDER Back End

- RESTful interface using fetch
- Node.JS server to handle the request
- Using MONGODB from node
Next Time

- Poster Session