Web Applications Outside Your Browser

CS-554 – WEB PROGRAMMING

Why?

GUI Programming

GUI Programming, in general, is cumbersome to develop, full of state issues, and an awkward field to develop in.

- Each environment has its own issues, limitations, and so on
- Usually have to write a different series of GUI code for different platforms

Web applications act as GUIs for browser based applications.

Web Applications without the Web

Traditionally, we view the web as something we load up on browsers on laptops, desktops, and mobile phones.

Nowadays, we can actually develop web applications that are not on browsers (as we know them) but instead appear as actual apps!

There are a shocking number of benefits to this:

- Can write an app in HTML, CSS, and JS; easy to use, familiar technologies that we know how to make apps in
- Can usually deploy a particular version of the browser with the app you're deploying, which allows you to develop towards a controlled environment.
- Can test that particular version of the browser thoroughly.

Deploying web apps as desktop applications

Many people use a node module called *electron* to wrap a node package up as an executable app that can be installed and run on desktops / laptops.

Electron will allow you to define a version of chrome that will load up a browser that is wrapped inside of an application.

More importantly, it allows you to run node code in your application.

This allows you to:

- Have a single page app that can access all your node code, such as reading and writing from files
- Setup menus, file saving windows, file open windows, access the clipboard, add localization, measure power, access the tray, etc

Deploying web apps as mobile apps

It is also possible to make HTML, JS, and CSS apps that get wrapped up as a mobile phone application that can be added to the app store.

A very popular software to do this is *Cordova*.

Cordova exposes native phone functionality through the use of plugins that are accessible in your JavaScript code, such as working with files, taking pictures, tracking geolocation, and so on.

Electron

What is Electron?

Electron is a framework to allow you to write GUI Applications that can execute node functionality.

It allows you to serve static HTML and CSS files in the form of a desktop application that have access to Node features.

We will be modifying the electron app from previous lectures to be a single page application that does not use server information, but rather file information!

- https://electron.atom.io/

Using Electron with a React App

Setting up our previous book reader React App required a few minor changes.

- We had to remove the dependeny on *create-react-app* and convert it to a fully configured react application
- We had to tweak the configuration to allow it to run in an electron enviornment.
- We are now setting webpack dev server to run via the main node script, rather than command line
- Once the webpack dev server starts up, we then spin up an Electron window
- We then point the Electron window at the dev server.

Using native node modules in our new, React + Electron app

By running our app with the *electron* command, rather than the *node* command, and configuring webpack extensively, we are able to expose native node modules into our react app.

You can see, we removed our old react data file and are simply referencing our node data module that was previously on our server.

- All config and setup is done for you by cloning this lecture’s codebase.
- https://electron.atom.io/docs/tutorial/debugging-main-process-vscode/

Using Electron API

Electron provides a set of tools that you can use to make interaction with your file system incredibly robust.

Electron exposes these tooks in 2 APIs:

- One API to the main process (the node script that control the window, and rest of the app)
- One API to the renderer (the frontend portion of your application).

You can find a great deal of info on all these APIs on the Electron documentation

- https://electron.atom.io/docs/

Guided walkthrough

We will be adding some more Electron capabilities to our app!

We add a button to the Single Book Container that, when clicked, will copy the text of the book to your clipboard.

- https://electron.atom.io/docs/api/clipboard/

We will also add to our custom electron menu!

- https://electron.atom.io/docs/api/menu/
- We will add the ability to go directly to the book list by loading the book list URL on click.

Finally, we will make a new view that will allow us to create a new book.

Hybrid Mobile Applications

What are mobile applications?

Mobile applications are an incredibly broad field in the modern era. In general, they are anything that can be run on a mobile operating system

- Games
- Content Readers
- General Utilities
- Social Media

For the most part, anything can become a mobile application, and at this point, there's a mobile application for just about anything

There are three types of mobile applications

- Native App
- Hybrid App
- Web App

Native Mobile Applications

Native Mobile Applications are mobile applications that have been programmed using SDK tools in order to be built and deployed and run using instructions native to a mobile operating system.

Writing a mobile application as a native mobile application have a number of benefits:

- Performance is at its greatest with native applications
- Highest degree of control over how things work when writing native
- Can write more complex applications with respect to backend logic

However, there are downsides of native mobile applications:

- You have to write one application per operating system
- UI code is always difficult to write with non-UI oriented systems; programming languages are just not good at writing clean UI code!
- You have to hire developers proficient in developing in that particular OS with a limited set of languages

Web Applications on a Mobile Phone

Web Applications on mobile phones are any web application that is accessed while on a mobile device. They are transmitted via the internet, and they are stored on an external server.

There are a few benefits of simply having users run your web app on their phone:

- Never have to deploy the app; it can be updated just by uploading a new file to your server
- Won't have to develop two applications; instead, you'll just be using media queries to make your mobile display smaller.
- Easy to develop a UI in HTML and CSS

However, there are downsides of relying on users to run your web app over the internet:

- They have no access to most native OS calls
- Users could lose internet, have low service, etc; what then?
- Have to keep requesting your assets.

Hybrid Mobile Applications

Hybrid Mobile Applications are mobile applications that are bundled up, built, and deployed to mobile applications. They are written with HTML, CSS, and JS driven UIs but interact with native operation system functions. A Hybrid Mobile Application is usually a UI run in a WebView.

Hybrid Mobile Applications have some profound benefits:

- Can be thought of as network-optional web applications
- Most logic used to build out a web application can be directly ported to the mobile application.

However:

- Not as performant as mobile applications
- Have to keep deploying changes through application stores.
- If you can’t find a plugin to perform some functionality, you have to write one yourself in every language that your mobile app should be deployed to.

How are hybrid mobile applications related to web development?

Hybrid Mobile Applications tend to be web applications that are not run off web servers.

- Web application companies tend to make hybrid mobile applications that share some of their codebase with their web application
- Tend to be developed with some form of SPA framework in order to emulate the concept of navigation
- Tend to be highly cross-platform

Most skills and tools you use as a web developer are used as a hybrid mobile application developer, since you are writing a UI in HTML, CSS, and JavaScript.

- Major difference is that, using JavaScript, you have the ability to interact with the native system calls through the use of plguins
- Unlike Electron, you cannot directly call filesystem code from your frontend code.

What technologies do web dev and hybrid mobile application share?

Most of the technologies you use in web application development appear when building hybrid mobile applications:

- HTML
- CSS
- JavaScript
- SASS to build CSS
- WebPack and other task runners

For the most part, any plugin or tool you would use on your web application can be easily dropped into your application.

Hybrid mobile application technologies

Framework	Description
Cordova	A tool to create and bundle web applications as mobile apps with access to system functions through JavaScript methods.
Phonegap	An Adobe productized version of Cordova
Ionic	A framework built on top of Cordova for building and deploying apps with standardized views and an APIs for common view-related tasks
Sencha Touch	An MVC JS Framework for building out mobile applications

Apache Cordova

What is Apache Cordova?

Apache Cordova is an open source hybrid mobile application development framework that allows for you to program cross platform mobile applications.

Cordova is first and foremost a command line application that allows you to create projects that can be built into hybrid mobile applications. Cordova applications are built out as webviews with a UI created with HTML and CSS, however it also provides access to native system methods through calling JS methods that will activate native code.

Cordova no longer just handles mobile applications, but rather supports:

- Android
- iOS
- Windows 8.1
- Blackberry 10
- Ubuntu
- Browsers

Native Functionality in a Cordova Application?

Cordova based applications expose native functionality through the use of plugins.

There are two components to each plugin:

- Native code written to perform the work needed
- JavaScript code exposing an API to interact with each plugin.

From the perspective of your frontend code, Cordova plugins are inherently asynchronous

- The webview runs on one thread, while the native code runs on another; there is IPC at work when using the JS plugin
- The JS code has no easy way of determining how much work is going on, is left, is complete besides the use of events from the plugin's native cde.

How do we make Cordova Plugins?

The actual work for Cordova plugins are written in the native code for a particular mobile operating system.

- Java for Android
- Swift or Objective C for iOS
- C# for MS

Plugins do not need to support all Operating Systems that Cordova supports, however you cannot use a plugin on a system that the plugin does not support.

A complete guide for developing a plugin can be found on the Cordova documentation

- https://cordova.apache.org/docs/en/latest/guide/hybrid/plugins/

Cordova Plugins: Not just for native functionality

Mobile applications often have some sort of special algorithmic code that is not easy or practical to write using JavaScript code that runs other Cordova Plugins, making it practical to write a custom Cordova Plugin to implement code that is specific for your application.

While your UI and a bulk of your application logic would still be written in HTML, CSS, and JS you would gain the ability to write code that runs on a separate thread for these potentially complicated algorithms. You also are still able to write the majority of your application in cross-platform friendly webview code, while writing minimal amounts of platform specific code.

One such example of this need is writing custom synchronization code; you may want to write the synchronization logic in your native application code, rather than run through a great deal of JS that has to constantly scan through your phone's storage in order to see what's been synced or not.

Ionic

While Cordova allows you to take a series of HTML and CSS and JavaScript files and have your phone load these files up, it does not provide you with any particular boilerplate to help get you started.

Instead, many developers prefer to use a framework such as Ionic.

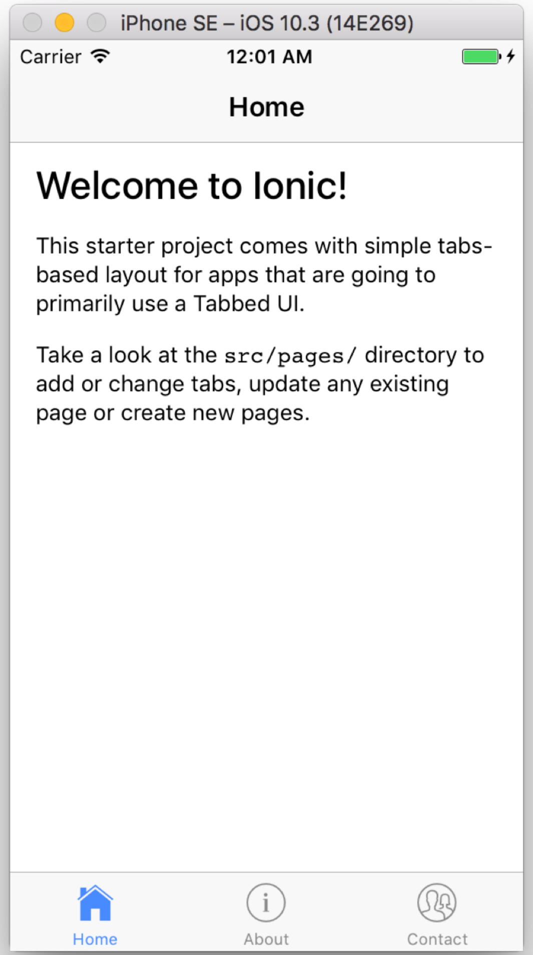
Ionic wraps around Cordova, and also provides a setup of Angular JS.

- https://ionicframework.com/

A Brief Demo

```
npm install -g ionic cordova

ionic start cutePuppyPics

cd cutePuppyPics

ionic run ios
```

This will create a basic React Native ios package, and will allow you to have many cool features such as live reloading changes. You can edit index.ios.js and watch the changes occur on your simulator!



Welcome to Ionic!

This starter project comes with simple tabs-based layout for apps that are going to primarily use a Tabbed UI.

Take a look at the `src/pages/` directory to add or change tabs, update any existing page or create new pages.

React Native

What is React Native?

“React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components.

With React Native, you don't build a “mobile web app”, an “HTML5 app”, or a “hybrid app”. You build a real mobile app that's indistinguishable from an app built using Objective-C or Java. React Native uses the same fundamental UI building blocks as regular iOS and Android apps. You just put those building blocks together using JavaScript and React.”

- https://facebook.github.io/react-native/
- Those descriptions from the React Native page

JavaScript but not a hybrid?

React Native runs JavaScript without running a webview. It accomplishes this by going a level further, and using your device's native JavaScript Environment to handle the actual application logic..

Your JSX will create JavaScript that calls Native code, to build your UI. In a small way, this is similar to how Cordova runs Native code, as well.

To note, not all your usual browser APIs are available in React Native, since there are no windows or browsers being run!

- https://facebook.github.io/react-native/docs/javascript-environment.html

Paradigm Reversal

React Native takes the Hybrid Mobile App paradigm and reverses it.

The usual hybrid mobile application paradigm is to display an interface from a webview, where the display is from HTML, CSS, and JavaScript.

React Native's approach is to use JavaScript to handle the component based logic, such as what components get loaded and how to handle property changes, but instead of using React-DOM to render HTML that is styled with CSS, it renders Native Components.

It is still entirely possible to still use familiar libraries, such as React Router and Redux

- https://reacttraining.com/react-router/native/guides/quick-start
- https://github.com/reactjs/react-redux

React Native… Desktop?

Fundamentally, React is just a component library. As such, React Native is able to rely on the same paradigms simply by accounting for being run in a native environment.

Naturally, soon after, people began to realize that you can use this same concept to apply to writing native MacOS apps, and native Windows apps as well!

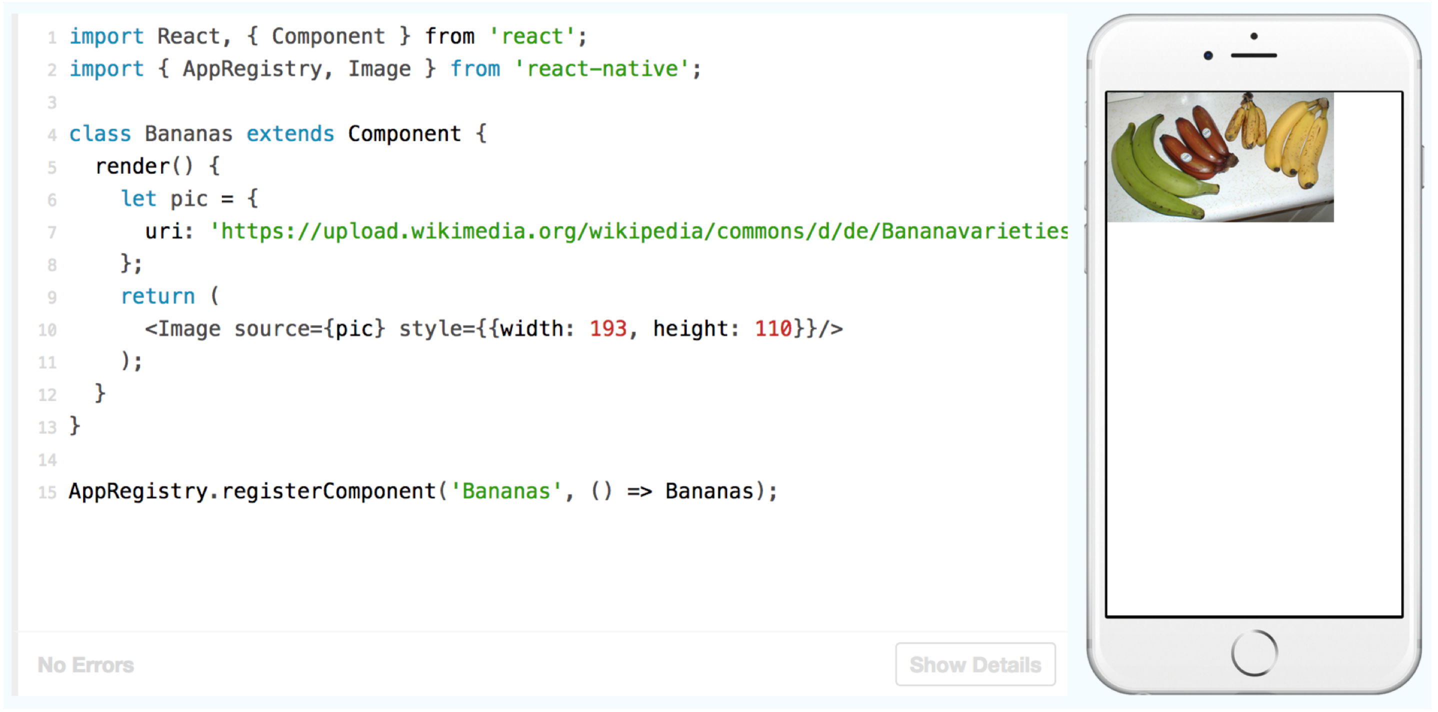
The paradigm is fairly solid, and it can be expected that this trend will only continue in the future.

- https://github.com/ptmt/react-native-macos
- https://github.com/Microsoft/react-native-windows

What does React Native look like?

React Native code, shockingly, looks exactly like regular React Code!

The magic comes in Image being a Native Component.



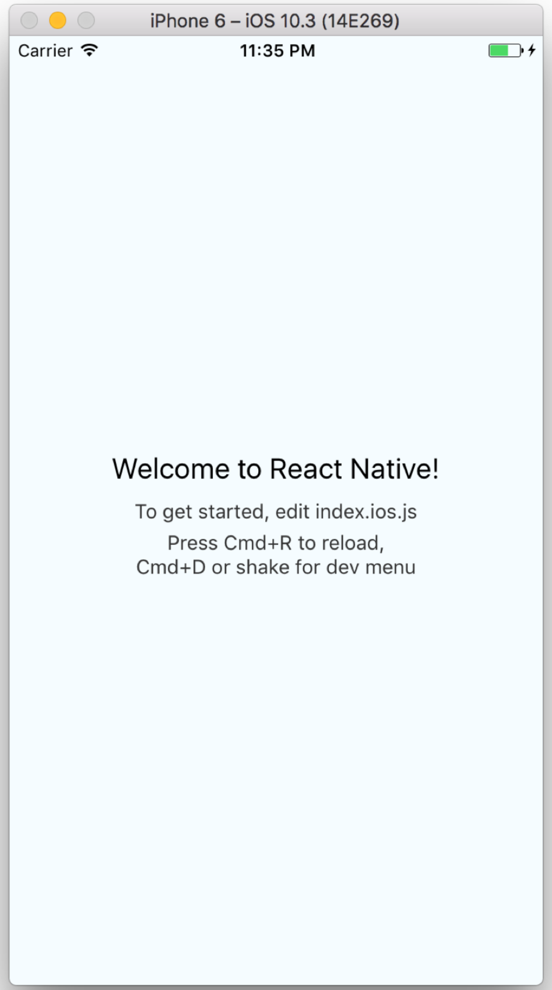
```
import React, { Component } from 'react';
import { AppRegistry, Image } from 'react-native';

class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}

AppRegistry.registerComponent('Bananas', () => Bananas);
```

No Errors

A Brief Demo

```
brew install watchman

npm install -g react-native-cli

react-native init AwesomeProject

cd AwesomeProject

react-native run-ios
```

This will create a basic React Native ios package, and will allow you to have many cool features such as live reloading changes. You can edit index.ios.js and watch the changes occur on your simulator!



Welcome to React Native!

To get started, edit index.ios.js

Press Cmd+R to reload,
Cmd+D or shake for dev menu