React.JS

Code 401 (Class 26)

A Bit of History

Typical Web Apps …

PHP, Java, Perl, Express, Meteor, Ember, etc.

1. Browser makes a request
2. Server handles it …
    a. Does a database lookup
    b. Does some operations
    c. Creates some markup
3. Sends Assets to the browser
    a. Markup
    b. CSS
    c. JS
    d. Images

Advantages

1. Well Understood
2. SEO

Disadvantages

1. Vertical Scaling
2. Unclear SOC

Modern, Single Page Web Apps

Same Core Paradigm, but much more service based

1. Browser makes an initial request
2. Server sends out minimal assets
   a. Shell Page
   b. Components
   c. JS
   d. CSS
3. As user interacts …
   a. Components request data (via REST, using ajax)
   b. Merge with templates
   c. DOM Re-Draws itself
   d. Feels like an app (No reloads)

Advantages

1. SOC
2. Horizontal Scaling

Disadvantages

1. SEO

Front End Code

Typically “Browser Code”, but it’s really “User Interface”

Could be a browser app, phone app, terminal chat interface

Build UIs for REST APIs

ReactJS

How is it different than jQuery?

- No direct DOM manipulation (React does it, we do not)
- Where jQuery is largely procedural (and event driven) …
- Modern stuff like react is “Declarative Programming”
  - State what you want to achieve (React), not how you want to achieve it (jQuery)

in jQuery — “go to the dom, find this thing, and on some event do this thing to it”

in React — “when a user is logged in, i want a nav bar”

There are still some procedural bits to react that react to events, but the UI is constructed declaratively

React is template-ing + state management…similar to handlebars, but with state

As the state changes, the template is automatically re-renders (the view updates itself)

Application Parts

Scaffolding

- Node/package.json
- Babel (Transpiler)
- Webpack (Bundler)

Back End Coding

- RESTful Services
- Databases
- Security & Access

Front End Coding

- Components
  - Markup
  - Behaviors
  - Styling

The Stitching & Delivery of a React App

- You write your app and component code
- Babel will transpile your JS code into ES5
- WebPack will:
  - Compile your less or sass into CSS
  - Assemble your imported dependencies in the right order
  - Create one or more “bundles” of code (JS/CSS)
- Your node server will send the bundles and your base app
- The browser will render the app and initial state
- Subsequent requests only go to the server for data

What is a component?

Simply put, a React component is a thing that can be **independently** rendered in the browser

- Carries it’s own Markup, CSS, Javascript
- Has behavior
- Might have state

Components are the core of both User Interface and Experience

UI - The actual things a user interacts with

UX - The way they interact with them (workflows, behaviors, etc)

If it can stand alone, it’s a component.

Building a React App - Core Dependencies

webpack - The Bundler

webpack-dev-server - Sorta like nodemon — will reload the browser

html-webpack-plugin - Will generate the right (dynamic and different) script and css tags for the index.html file

node-sass -

sass-loader - Turns SASS into CSS

resolve-url-loader - Allows relative paths in the SASS code

css-loader - Turns CSS code into a JS object that webpack can understand

extract-text-webpack-plugin - Takes CSS code and turns it into a separate css bundle

babel-core - Transforms ES6 code into ES5

babel-loader - Hooks babel up to webpack

babel-preset-react

babel-preset-env

babel-plugin-transform-object-rest-spread (gives us the … for objects)

react - Front End Magic (actually the only thing that'll end up in the browser (aside from assets)

rect-dom - Turns react into a **browser** rendering engine

There's others ... react-native for devices, react-ncurses for terminal apps

Building a React App - Scaffolding

Webpack

Bundling Rules

webpack.conf.js

- Entry
- Output
- Plugins (add feature)
- Loaders (transforms files)

https://webpack.js.org/concepts/

Babel

Transpilation Rules

.babelrc

- plugins
- Presets (plugin bundles)

https://babeljs.io/

React App: Build and Deploy

index.html Container

bundle.js Components, Behaviors

bundle.css Layout and Style

As we go on, we'll be building huge apps with many more, but they'll always pack down to those 3 above.

Thinking in Modern UI (React/Angular/Vue)

Take a bigger picture look at things ...

- What is the basic visual shape of the site
- What are the user workflows
    - What components do we need to achieve?
    - How can we best break them down?
- What data to they require to render?
- What data needs to be persisted back to the server?
- How will they pass data back and forth (API, Service)?

https://reactjs.org/docs/thinking-in-react.html