Intro to React

JS Powered Templates & User Interfaces

Building UIs in Javascript

- Lately, the idea of server-side templating has gotten less popular, in favor of making our templates in Javascript
- Going back to the server for every change is cumbersome
- Extremely hard to have a JS-enhanced frontend that's in sync with the server
- Instead of sending over full HTML pages, sending a bare-bones pages that later gets populated with content from JS

A Few Different Options

- Like most things, a few options have emerged:
  - Google's **Angular** framework, which uses custom attributes to enhance HTML
  - The community made **Vue**, which allows you to write template-style HTML that's replaced with Javascript values
  - **Meteor**, which combines your frontend and your backend to maintain consistency
  - But the one we'll be learning is **React**

What is React?

- React is **declarative**, meaning it renders your data exactly as instructed, the same way every time
- It's built around combining separate isolated **components**, which promotes reusability and third party integrations
- It has a strict **data flow** for components, which means you always know where the data that was rendered came from, and it's always consistent
- It uses **DOM diffing** to only update what you changed, making your code much faster than traditional JS frameworks
- It promotes the philosophy of **learn once, write anywhere**
    - Once you learn React, you can use it to write web apps, iOS and Android apps, and more

Why Choose It?

- It has a ton of community support behind it
  - 5 million+ downloads a month
  - 30k+ npm packages with "react" in the name
- It is used by some of the largest companies
  - Facebook, AirBnB, Instagram, Dropbox, Netflix
- Once learned, it's *much* easier to drop into a codebase that uses it
  - Translated: It's much easier to get *hired* if you know it
- It is, in this instructor's opinion, the best framework

Enough Talk, Let's Get Started

https://github.com/wbobeirne/nycda-basic-react

Example Code - `index.js`

- This is the entry point for our webpack build
- All it does is it mounts our react component to the DOM
- This is a typical pattern, we don't want our React code to be care about where it lives, or how it's attached to the page
- We use `React` to render the component, and `ReactDOM` to handle the DOM mounting

Example Code - `App.js`

- `App.js` is a **react component**, a self contained piece of code that renders some DOM elements
- Components are classes that extend `React.Component`, and receive some methods as a result of it
- The main one used here, `render`, is called every time we need an up-to-date version of the component's view
- Components have many other important functions and attributes, but they can be as simple as one render function

Improving React with JSX

- We could continue to learn react using many more `React.createElement` calls, it would be lengthy and cumbersome
- Most people agreed that this was an annoying abstraction of the DOM, as opposed to HTML which is much more succinct
- What we want is a Javascript aware version of HTML that can live alongside our code
- And we can get this by using **JavaScript XML**, or JSX, a standard invented for React

JSX Example

```
// JS-only React
return React.createElement("div", { className: "app" },
    React.createElement("h1", { className: "app-title" }, "Hello!"),
    React.createElement("p", { className: "app-text" }, `
        This is ${libName}. Even though we don’t have any elements on the page
        to start, ${libName} quickly fills in the javascript content.
    `),
);

// JSX React
return (
    <div className="app">
        <h1 className="app-title">Hello!</h1>
        <p>
            This is {libName}. Even though we don’t have any elements on the page
            to start, {libName} quickly fills in the javascript content.
        </p>
    </div>
);
```

JSX Setup

- JSX is handled using a Babel preset called "react"
  - Install the node module `babel-preset-react`
  - Add it to the plugins array in .babelrc
  - This just transforms the JSX code we saw to the JS only code from before
- Next we'll want to install a syntax highlighter that understands JSX
  - In atom, install the plugin `language-babel`
  - In sublime, install the plugin `Babel`
- Finally, you'll want to change your eslint plugin to use the one in the project
  - Normally we don't use our own `.eslintrc`, we use one provided by a project
- Now that you're all good to go, let's convert `App.js`

Making Components

- One of the strengths mentioned about React was components
- Each component can only output **one** element
  - That element can have as many children as you want, and be as big as you want though!
- We'll demonstrate that by converting our App component into using components
- Let's make `components/Title.js` and `components/Description.js`
- Once that's done, we can require those two components in `App.js` and render them using JSX as well

Making Components (Code)

```
// At the top of your file
const Title = require("./components/Title");
const Description = require("./components/Description");

// ...later in render()
return (
    <div className="app">
        <Title />
        <Description />
    </div>
);
```

Passing Arguments to Components

- Most components don't just simply render statically defined content though
- We'll often want to pass arguments to them
- The arguments we pass are called `"props"`, or properties
- The same way we give HTML elements properties, we can give components properties
- Let's pass some content inside the `<Title>` tag for its title
- And let's give the `<Description>` tag a `library` property

Passing Arguments to Components (Code)

```
// App.js - render()
return (
    <div className="app">
        <Title>Hello!</Title>
        <Description library="react" />
    </div>
);
```

```
// components/Title.js - render()
return (
    <h1 className="app">{this.props.children}</h1>
);
```

```
// components/Description.js - render()
const { library } = this.props;
return (
    <p>
        This is {library}. Even though we don't have any elements on the page
        to start, {library} quickly fills in the javascript content.
    </p>
);
```

Properties Explained

- Properties are arguments that components receive when used
- They are accessible at the `this.props` object
- Components **cannot** change their properties, only the parent can
- Having a component's properties changed causes it to update
- Content inside of a jsx tag is sent as a special `children` property
- Otherwise, all properties are sent by the named tags in jsx
- We will be looking at properties more closely in future lessons

Additional Reading

- React - Documentation
- React - Introducing JSX
- React DevTools