React 1: Create React App & JSX

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session’s Content

- Heroku deployment
- Heroku PostgreSQL

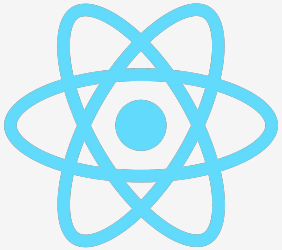
Today’s Content

- React
  - Overview
  - Node.js installation
  - NPM (Node package manager)
- Create React App
- JSX
  - Embedding expressions
  - Attributes
  - Injection attacks
  - Objects

React

Overview

- Open-source JavaScript library for creating interactive user-interfaces
- Maintained by Facebook & an open-source community of developers
- React is only concerned with rendering data to the Document Object Model (DOM)
- Additional libraries for state management & routing are required
- Recommended toolchains:
  - Single-page application - Create React App
  - Server-rendered website - Next.js
  - Static content-oriented website - Gatsby
- Alternatives:
  - AngularJS - https://angularjs.org
  - Vue.js - https://vuejs.org
- Resource: https://reactjs.org



Node.js Installation

- Open-source JavaScript runtime environment
- Used on both the client & server side
- Operates on a single-thread event loop, using non-blocking I/O calls
- Resource: https://nodejs.org/en

NPM

- Node package manager
- Pre-installed package manager
- Installs Node.js programs from the npm registry
- Resource: https://www.npmjs.com

Create React App

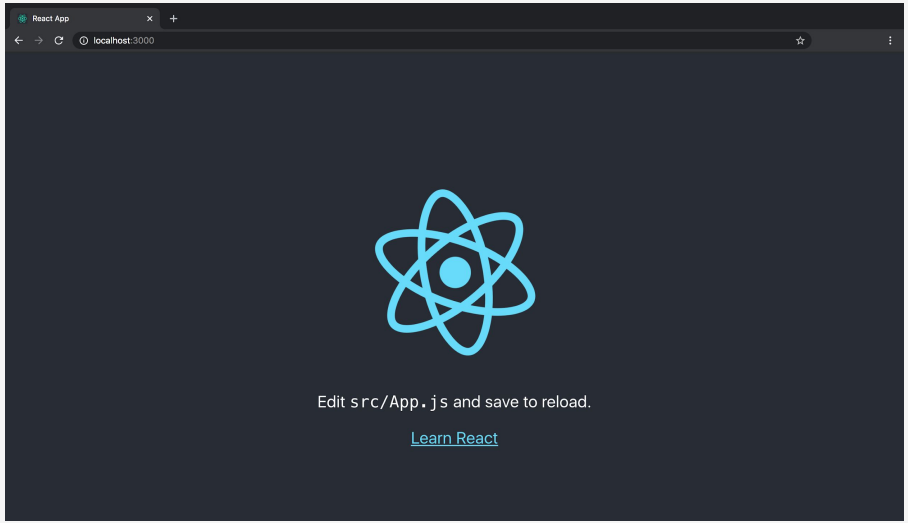
Create React App

- Create React App is the best way to start building a new single-page application in React
- What is a single-page application or SPA?
  - Dynamically rewrites the page with new data from the web server
  - HTML, JavaScript & CSS is either retrieved by the browser with a single page load or dynamically loaded & added to the page in response to user actions
- To create & start a new single-page application using Create React App, run the following commands:

```
npx create-react-app dog
cd dog
npm start
```

Create React App

- The development server will start & navigate you to http://localhost:3000



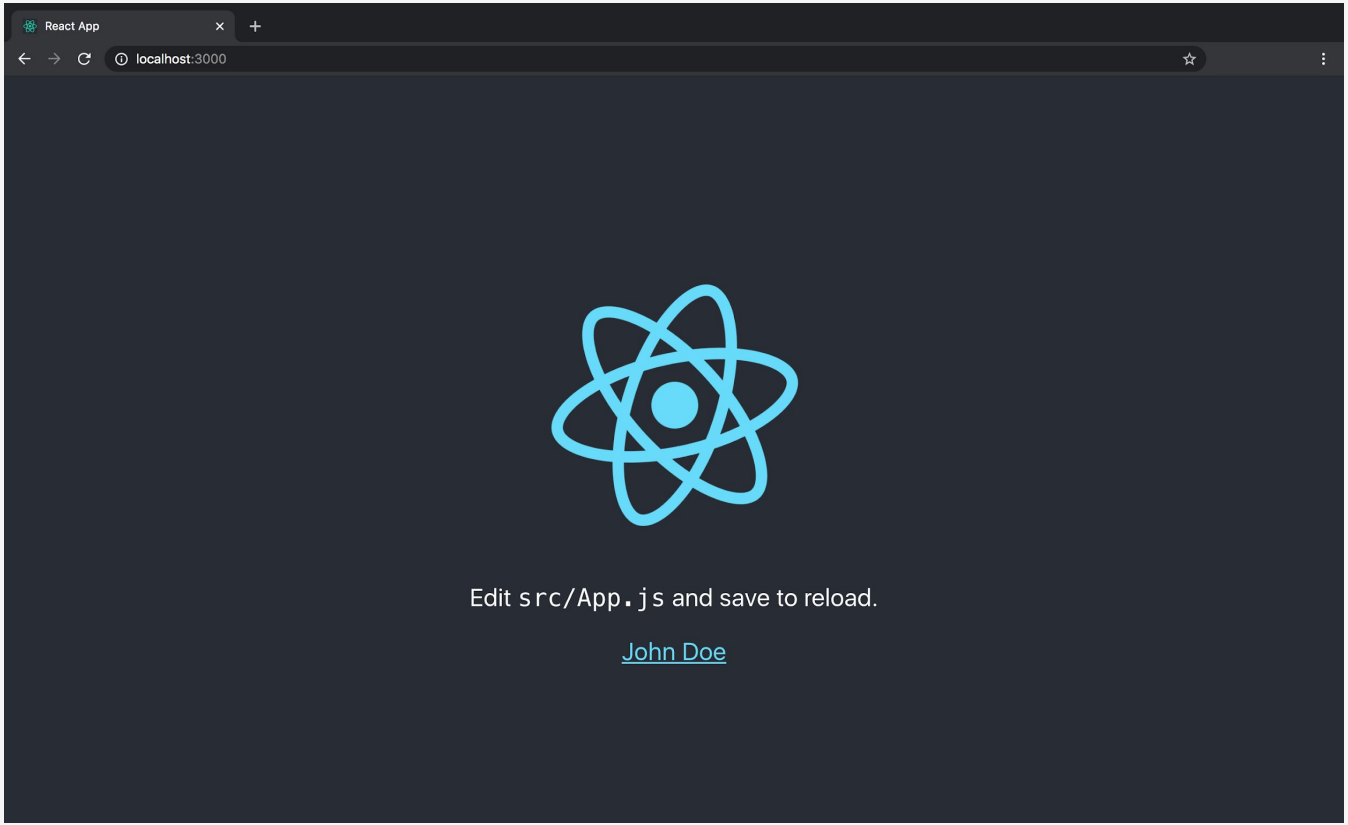
File Structure

- App structure:
  - `node_modules/`
  - `public/`
  - `src/`
  - `.gitignore`
  - `package.json`
  - `package-lock.json`
  - `README.md`

File Structure

- Expand `public/` & `src/` directories
- In `public/`, read the comments in `index.html`
- In `src/`, understand the purpose of each file & how they interact with each other
  - `App.css` - `css` styles specific to `App.js`
  - `App.js` - each component should have its own `css` & `test.js` file
  - `App.test.js` - tests specific to `App.js`
  - `index.css` - global `css` styles
  - `index.js` - Entry point. This file interacts with `public/index.html`
  - `serviceWorker.js` - required for Progressive Web Apps. You will learn about PWAs in **Adv App Dev Concepts**
  - `setupTests.js` - we will look at this is in **React 9: Automation Testing**
- In `src/App.js`, change the text **Learn React** to **<your name>**
- The development server will reload the browser when changes are made & saved
- Navigate to http://localhost:3000
- A sample React application has been provided in this directory. Files unrelated to this topic have been removed

Create React App



File Structure

- React does not care how CSS styles are defined, though, CSS classes perform better than inline styles
- Best practice is to define your styles in a separate CSS file & reference them using the `className` property
- In `src/App.css`, remove all styles except `body` & declare the following style:

```
.container {
  display: flex; /* Creates a flex container */
  flex-direction: column; /* Vertical flex direction */
  justify-content: center; /* Items centered vertically */
  align-items: center; /* Items centered horizontally */
}
```

File Structure

- In `public/index.html`, replace the current HTML with the following:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="description" content="Dog application created using create-react-app" />
    <title>Dog</title>
  </head>
  <body>
    <!-- Alternative content displayed to a user who has disabled <script> in their browser or has a browser that does not support <script> -->
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <!-- Everything in the "root" DOM node is managed by React DOM. React applications usually have only one "root" DOM node -->
    <div id="root"></div>
  </body>
</html>
```

JSX

JSX

- Is the following variable declaration HTML? `const element = <h1>Hello, World!</h1>`
- No, this is a syntax extension to JavaScript called JSX
- A semicolon at the end of the JSX is optional
- Using JSX with React to describe what the UI should look like is recommended
- You can think of JSX as a template language, but with the full capabilities of JavaScript
- React separates concerns with loosely coupled units of JSX called components. We will look at this in the next session

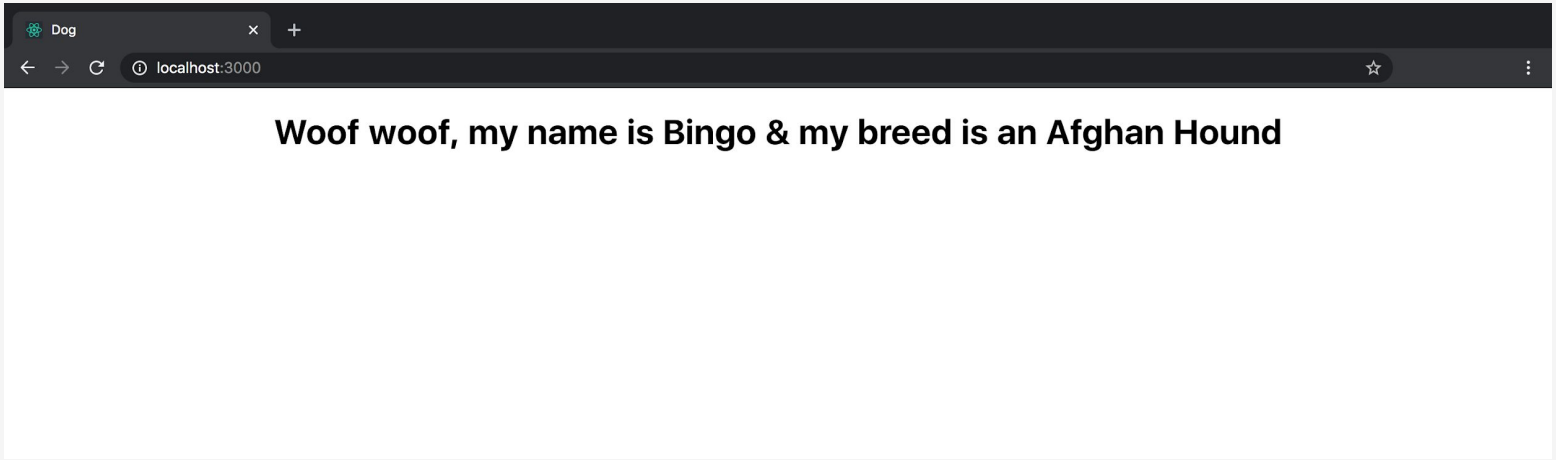
Embedding Expressions

- In `App.js`, update the code to the following:

```
import React from 'react'

const dog = {
  name: 'Bingo',
  breed: 'Afghan Hound',
}

function formatDog(dog) {
  return `Woof woof, my name is ${dog.name} & my breed is an ${dog.breed}`
}

function App() {
  return (
    <div className='container'>
      <h1>{formatDog(dog)}</h1>
    </div>
  )
}

export default App
```

- What is happening?
  - Declare a function called `formatDog()` which accepts one argument called `dog` & returns the `dog name` & `breed`. **Note:** template literals are enclosed by backticks instead of double or single quotes. Placeholders are indicated by the dollar sign & curly braces
  - `formatDog()` is called in the JSX & accepts the `dog` object as an argument

Embedding Expressions



Embedding Expressions

- In `App.js`, update the code to the following:

```
import React from 'react'

const dog = {
  name: 'Bingo',
  breed: 'Afghan Hound',
}

function formatDog(dog) {
  return `Woof woof, my name is ${dog.name} & my breed is an ${dog.breed}`
}

function getGreeting(dog) {
  if (dog) {
    return <h1>{formatDog(dog)}</h1>
  }
  return <h1>Uh...who are you?</h1>
}

function App() {
  return <div className='container'>{getGreeting()}</div>
}

export default App
```

- What is happening?
  - Declare a function called `getGreeting()` which accepts one argument called `dog` & returns JSX if an argument is/is not provided

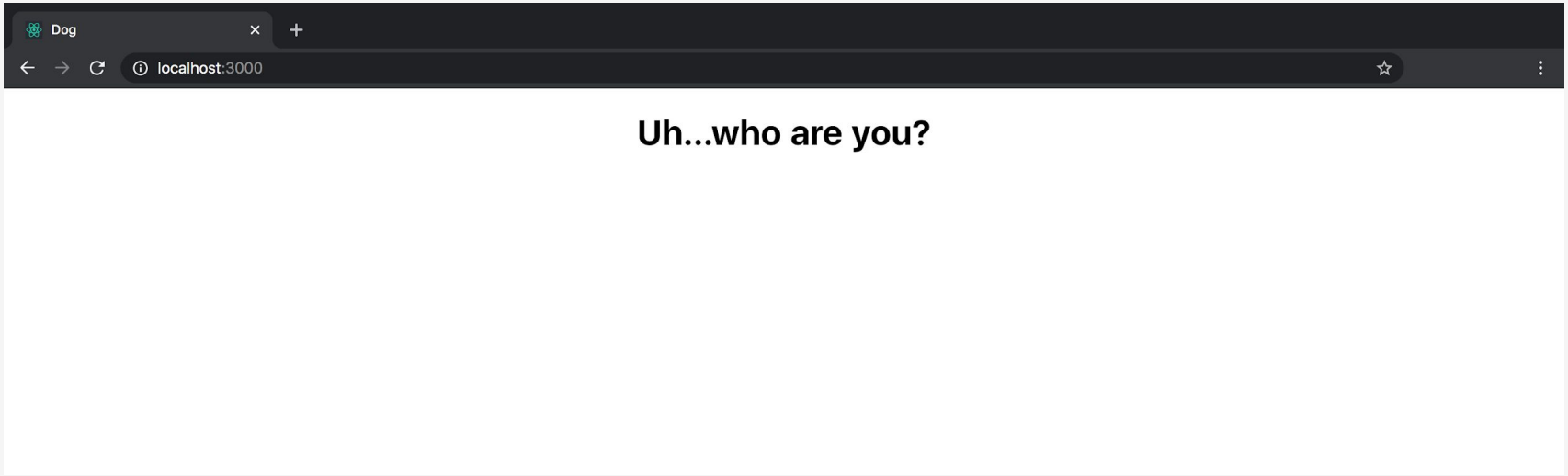
Embedding Expressions

- In `index.js`

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
)
```

- What is happening?
  - Render a React element into the DOM in the container, i.e., `root` in `/public/index.html` & returns a reference to the component, i.e., App

Embedding Expressions



Attributes

- In `App.js`, update the code to the following:

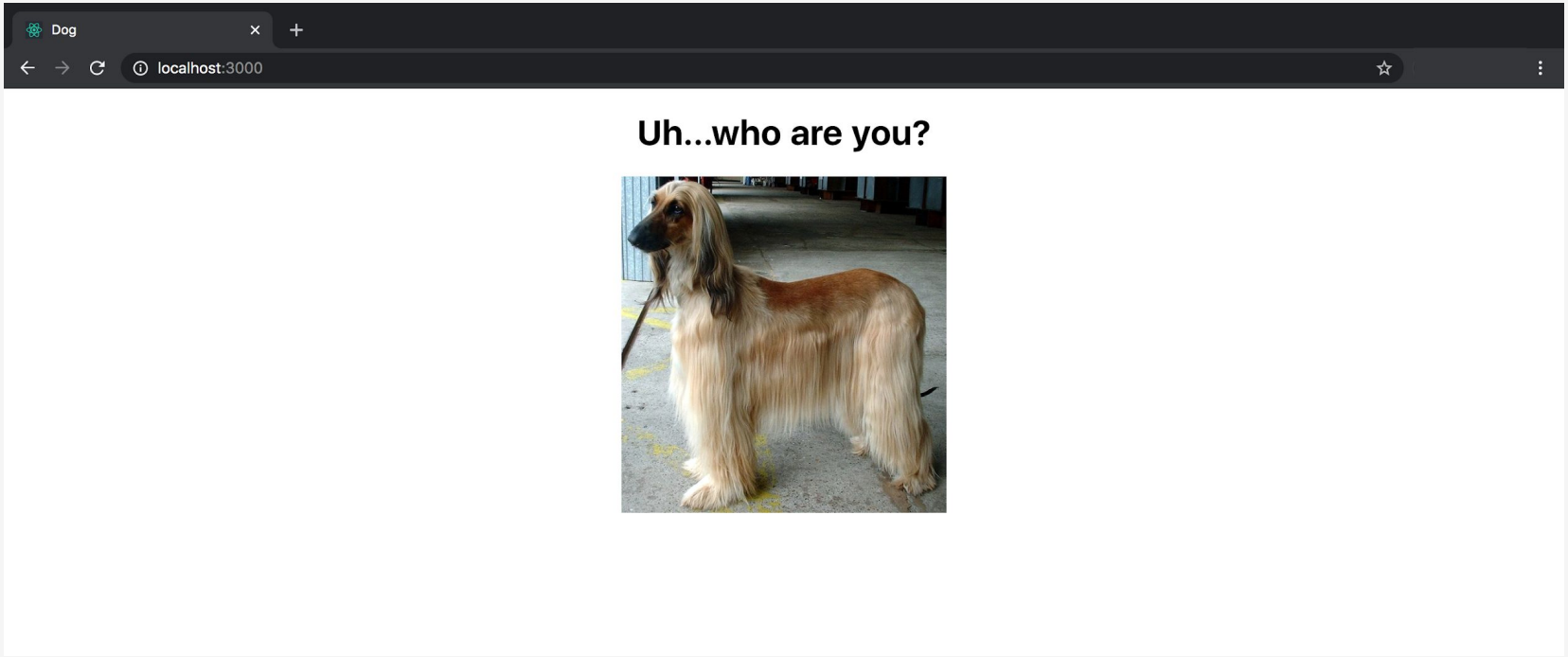
```
import React from 'react'
import afghanHoundImg from './img/afghan-hound.jpg'

const dog = {
  name: 'Bingo',
  breed: 'Afghan Hound',
  img: afghanHoundImg,
}

function formatDog(dog) {
  return `Woof woof, my name is ${dog.name} & my breed is an ${dog.breed}`
}

function getGreeting(dog) {
  if (dog) {
    return <h1>{formatDog(dog)}</h1>
  }
  return <h1>Uh...who are you?</h1>
}

function App() {
  return (
    <div className='container'>
      {getGreeting()}
      <img src={dog.img} alt='afghan hound' width='300' />
    </div>
  )
}

export default App
```

Attributes

- Quotes are used to specify string literals as attributes
- Curly braces are used to embed an expression in an attribute
- React DOM uses camelcase property naming convention, i.e., in JSX, `className` is used instead of the HTML attribute name `class`

```
<img src={dog.img} alt='afghan hound' width='300' />
```

Attributes



Injection Attacks

- Values in JSX are escaped by React DOM before rendering
- Ensures potentially malicious input can never be injected
- Everything is converted to a string before being rendered
- Helps prevent cross-site scripting attacks

Objects

- JSX is compiled down `React.createElement()` calls by Babel
- Babel is used to convert ES6 code into a backwards compatible version of JavaScript in current & older browsers, i.e., Internet Explorer
- Resource: https://babeljs.io

```
React.createElement(
  'div',
  {
    className: 'container',
  },
  getGreeting(),
  React.createElement('img', {
    src: dog.img,
    alt: 'afghan hound',
    width: '300',
  })
)
```

```
<div className='container'>
  {getGreeting()}
  <img src={dog.img} alt='afghan hound' width='300' />
</div>
```