Web Applications

CSE183

Fall 2020

Introduction to React

Baskin Engineering
UC SANTA CRUZ

Notices

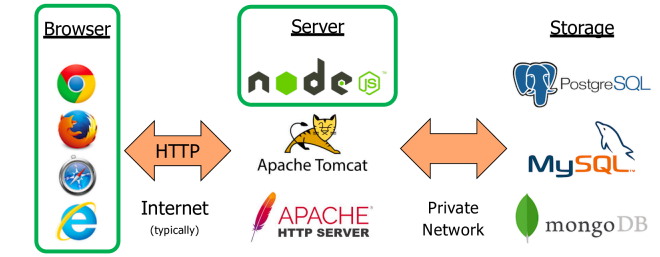
- **Assignment 4** due 23:59 **Thursday October 29**
- **Quiz 2** during class **Friday October 30**

Today's Lecture

- React Basics
- React Components & JavaScript XML (JSX)
- React Event Handling
- JSX Programming
- Component Lifecycle
- Stateless Components
- Component Communication

- Questions

Full Stack Web Applications



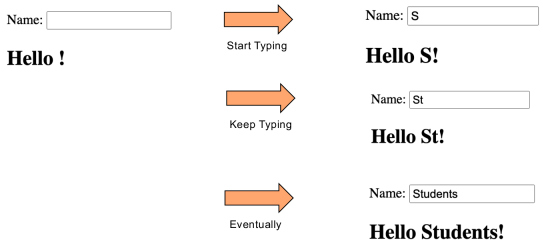
React Basics

- JavaScript framework for writing the web applications
  - Quick responses when running in browser
  - Less opinionated that other 4th Gen Web App Frameworks
    - Restricts itself to rendering views and handling user interactions
    - You can use any server and/or database you like ☺
- Uses the Model View Controller pattern
  - View constructed from Components
  - Supports HTML templating (optional, but heavily used)
- Minimal server-side support required
  - Most React apps can be deployed in any web server
- Intended for large and single page applications
  - Has modules, reusable components, testing hooks, etc.

Why "React"?

- Because it simplifies reacting to changes of state
- Side benefit:
  - Gives you the ability to keep application state away from the DOM
    - But *you* must code in a way that makes use of this

A Basic HTML / JavaScript App



A Basic HTML / JavaScript App

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 HTML Basic</title>
  </head>
  <body>
    <script type="text/javascript" src="basic.js"></script>
  </body>
</html>
```

```
const label = document.createElement('label');
const input = document.createElement('input');
const message = document.createElement('h2');

label.textContent = 'Name: ';
input.setAttribute('type', "text");
message.textContent = 'Hello !';

document.body.appendChild(label);
document.body.appendChild(input);
document.body.appendChild(message);

function handleInput(event) {
  message.textContent = `Hello ${event.target.value}!`;
}

input.addEventListener('input', handleInput);
```

Name:

Hello !

A Basic React App

We Write:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 React Basic</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

React Generates:

But How?

¯_(ツ)_/¯

Basic React App

Folders & Files

```
build  ( generated app ends up here )
public
  index.html
src
  index.js
  App.js
package.json
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 React Basic</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App/>, document.getElementById("root"));
```


Node.js Config: `package.json`

```
{
  "name": "cse183-react-basic",
  "version": "1.0.0",
  "description": "CSE183 React Basic",
  "author": "David Harrison <dcharris@ucsc.edu>",
  "license": "UNLICENSED",
  "repository": "none",
  "devDependencies": {
    "react-scripts": "^4.0.0"
  },
  "dependencies": {
    "react": "^17.0.1",
    "react-dom": "^17.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Meta-data for this JavaScript package

3rd party development tools

3rd party packages needed at runtime

Possible arguments to 'npm run'

Browsers we'd like the production version to work in

Browsers we'd like the development version to work in

React Components

src/index.js

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App/>, document.getElementById("root"));
```

src/**App.js**

```
import React from "react";

class App extends React.Component {
  state = {
    name: null
  };
  constructor(props) {
    super(props);
  }
  render() {
    ...
  }
}

export default App;
```

React Component Rendering

With Intermediate Variables:

```
render() {
  const label = React.createElement('label', null, 'Name: ');

  const input = React.createElement('input',
    { type: 'text', value: this.state.name });

  const message = React.createElement('h1', null,
    'Hello ', this.state.name, '!');

  return React.createElement('div', null, label, input, message);
}
```

Whenever `this.state.name` changes, content of the `message` element ***automatically*** changes ☺☺☺☺☺

Thank you React!

React Component Rendering

Without Intermediate Variables:

```
render() {
  return React.createElement('div', null,
    React.createElement('label', null,'Name: '),
    React.createElement('input',
      { type: 'text', value: this.state.name }),
    React.createElement('h1', null,
      'Hello ', this.state.name, '!');
    );
}
```


JavaScript XML (JSX)

- A syntax extension to JavaScript
- Recommend mechanism for describing UIs in React
- Akin to template languages
  - But it comes with the full power of JavaScript ☺
- *"This funny tag syntax is neither a string nor HTML"*

```
const element = <h1>Hello, world!</h1>;
```


https://reactjs.org/docs/jsx-in-depth.html

React Component Rendering

With JSX:

```
render() {
  return (
    <div>
      <label>JSX Name: </label>
      <input
        type="text"
        value={this.state.name}
      />
      <h1>Hello {this.state.name}!</h1>
    </div>
  );
}
```

React Event Handling Problem

```
class App extends React.Component {
  ...
  handleInput(event) {
    this.setState({ name: event.target.value });
  }
  ...
}
```

Consider:

```
<input type="text" onChange={this.handleChange}/>
```

Problem! `this` has no scope ☹

React Event Handling

Arrow functions to the rescue ☺

```
class App extends React.Component {
  ...
  handleInput = (event) => {
    this.setState({ name: event.target.value });
  }
  render() {
    return (
      <div>
        <label>JSX Name: </label>
        <input
          type="text"
          value={this.state.name}
          onInput={(event) => this.handleInput(event)}
        />
        <h1>Hello {this.state.name}!</h1>
      </div>
    );
  }
}
```


React & JSX Code Conventions

- Choice of word separator in multi-word variable name:
  - `camelCase` vs. `dash-case`
- JavaScript is case sensitive, but HTML is not
- JSX looks like HTML but is more like JavaScript
- Recommend using `camelCase` for attributes in JSX
  - E.g. `return <CustomButton color="fireBrick" />;`

JSX Programming Fundamentals

- JSX is parsed into `React.createElement()` calls
  - Very important concept, try to remember it
- `React.createElement(type, props, ...children);`
  - `type`: HTML tag (e.g. `h1`, `p`) or any `React.Component` sub-class
  - `props`: attributes (e.g. `type="text"`) (must be in `camelCase`)
  - `children`: Zero or more children which can be :
    - Strings
    - Numbers
    - React Components
    - An array of the above

JSX Template Restrictions

- JSX Templates ***must*** return a valid `children` parameter
- Templates can have JavaScript scope variables and expressions

```
<div>{foo}</div>
```

  - Valid if `foo` is in scope (i.e would be a valid `createElement() childen` parameter)

```
<div>{foo + ' and ' + bar()}</div>
```

  - Valid if `foo` & `bar()` are in scope

- Template must evaluate to a value

```
<div>{if (something) { ... } }</div>
```

  - Doesn't work because `if` has no value
  - Same problem with `for` loops
    - And any other JavaScript statement that does not return a value
- Leads to contorted JSX ☹
  - E.g. Anonymous immediate functions

```
<div>{ (function() { if ...; for ...; return val;})() }</div>
```


JSX Conditional Rendering

- Use JavaScript Ternary operator (`?:`)

```
<div>{this.state.french ? <b>Bonjour</b> : "Hello"}</div>
```

- Use JavaScript variables

```
let greeting;
const en = "Hello";
const fr = <b>Bonjour</b>;
let {french} = this.state;
if (french) {
    greeting = fr
} else {
    greeting = en
};
<div>{greeting}</div>
```

Much Better ☺

JSX Iteration

- Use JavaScript array variables

```
let lis = [];
for (let i = 0; i < data.length; i++) {
  lis.push(<li key={data[i]}>Data Value {data[i]}</li>);
}
return <ul>{listItems}</ul>;
```

- Functional programming

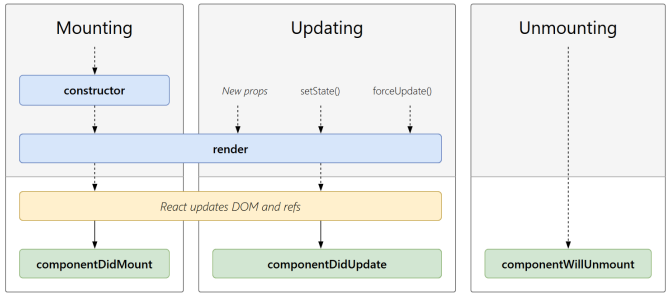
```
<ul>{data.map((d) => <li key={d}>Data Value {d}</li>)}</ul>
```


JSX Styling

```
import React from 'react';
import './MyComponent.css';

class MyComponent extends React.Component {
  ...
  render() {
    return (
      <span className="whatever">
      ...
     </span>
   );
  }
  ...
}
```

React Component Lifecycle



https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

React Component Lifecycle

```
class MyComponent extends React.Component {
  state {
    counter: 0
  }
  ...
  componentDidMount() { // Start 2 sec timer to increment a counter
    const incFunc = () => this.setState(
      { counter: this.state.counter + 1 });
    this.timerID = setInterval(incFunc, 2 * 1000);
  }
  componentWillUnmount() { // Cancel the timer
    clearInterval(this.timerID);
  }
  ...
}
```


React Stateless Components

- Components with properties but no methods can be functions rather than classes:

```
function MyComponent(props) {
  return <div>My name is {props.name}</div>;
}
```

- Or destructured:

```
function MyComponent({name}) {
  return <div>My name is {name}</div>;
}
```

- React Hooks
  - Add state to stateless components
  - https://reactjs.org/docs/hooks-intro.html

React Component Communication

- Parent to child:
  - Properties (attributes)

```
<ChildComponent param={dataForChild} />
```

- Child to parent:
  - Callbacks

```
this.parentCallback = (dataFromChild) => {
    /* process dataFromChild */
};
<ChildComponent callback={this.parentCallback}> />
```

- React Context
  - Global variables for subtree of components
  - https://reactjs.org/docs/context.html

Upcoming Lectures

- Wednesday: Single Page and Responsive Web Applications
- Friday: Quiz 2 & Assignment 5 Introduction

Tasks

- **Assignment 4** due 23:59 **Thursday October 29**
- Study for **Quiz 2** during class **Friday October 30**