Lab – Intro. to Web Dev with React

COS 420/520 - Introduction to Software Engineering

What Is This Lab For?

- Gaining a basic understanding of the tools and technologies used in web dev
- Gaining some experience in React
- Learning what options are available to use in web dev projects
- Learning what to Google once this lecture period ends

Front-end Development I

- The **front end** of a website is everything the user sees or interacts with, i.e.:
  - Graphic Design (GD, how the website looks - deals with colors, layout, fonts, etc.),
  - User Interface (UI, how the website feels - deals with logic behind buttons, animations, etc.), and
  - User Experience (UX, how users experience a product - deals with *everything* surrounding a user's interaction with something. Think: Apple's hardware, software, and services).

Front-end Development II

- Front-end development boils down to three file types that browsers can interpret:
  - Hypertext Markup Language (HTML, defines the content/format of a website),
  - Cascading Style Sheets (CSS, defines how the site looks), and
  - JavaScript (JS, defines any logic behind buttons, links, etc.).

Back-end Development I

- The **back end** or **server-side** of a website is the part that **users do not see** and that enables the functionality of the front end, i.e.:
  - Servers, which handle communication between the front end, databases, and other services,
  - Databases, which store and organize information, and
  - Applications, which provide various functions such as hosting a website or monitoring traffic.

Back-end Development II

- Back-end development uses languages such as PHP, Ruby, Java, Python, .NET, or Node.JS to:
  - Handle incoming/outgoing requests for static information
  - Store and retrieve data from databases
  - Generate new content
  - Modify server files
  - And more

Intro. To React

- React is a library that simplifies and enhances front-end web development.
  - Created by Facebook in 2011 and open-sourced in 2013.
  - Based on intuitive understanding of how applications look and function:
    - Applications are made up of components.
    - Components display information and respond to input.
- In React, components respond differently depending on the state of the application.

React Step 0a: Install NVM

- Install Node Version Manager (*https://github.com/nvm-sh/nvm*)
  - Use either the wget or curl commands listed there.
  - This will allow MacOS and Linux users to easily configure their Node version on a per-project basis.
  - Windows users can either use the Windows Subsystem for Linux, as described in the above link, or can use Node without NVM.
- Alternatively, ...

React Step 0b: Install Node.js And NPM

- Install Node.js and Node Package Manager
  - Via Installer: ***https://nodejs.org/en/download/***
  - MacOS:
    - Homebrew: brew install node
    - MacPorts: port install nodejs
  - Linux:
    - Ubuntu: apt install nodejs npm
- This installation alone will not allow for (straightforward) per-project Node versioning.

React Step 0.5: Configure NVM

- With nvm, you can easily configure what version of Node.js you are using at any given time. This can be helpful when trying to run code from older projects.
- nvm install node - Install the most up-to-date Node version
- nvm install 15.9.0 - Install the specified Node version
- nvm use node - Switch to the default Node version
- nvm use 15.9.0 - Switch to the specified Node version
- nvm ls - List installed Node versions
- nvm ls-remote - List all available Node versions

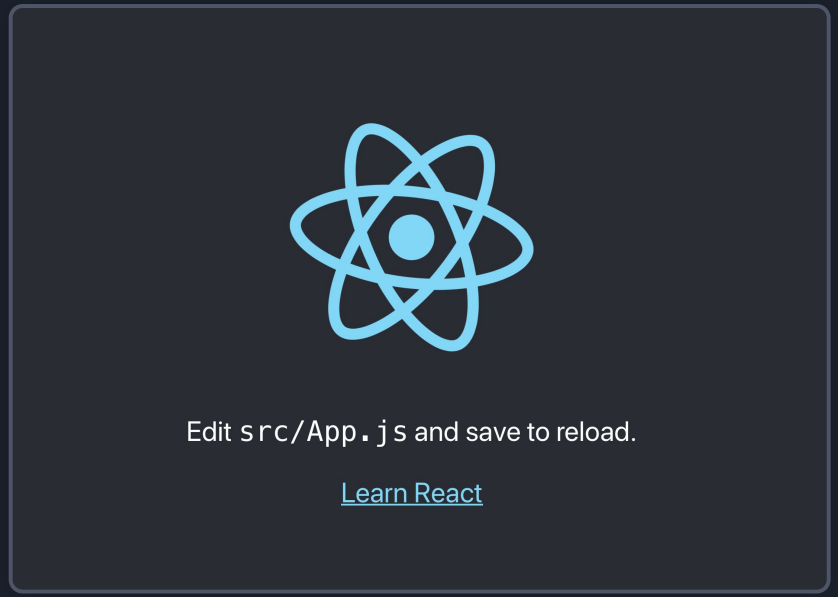
React Step 1: Create React App

- cd into an appropriate folder, then run npx create-react-app web-dev-lab

- Create React App (CRA) makes starting a new React project easy, but it should not be used for production versions of software.
  - There are many libraries included in CRA that you might not need.
  - Unneeded libraries create unnecessary security risk.

React Step 1: Start the Development Server

- cd web-dev-lab
- npm start or yarn start
- This will start a server at localhost:3000 (or the next available port).
- Use Ctrl + C to stop the server.
- A browser window will open to a page that looks like this:



CRA Folder Structure I

- node_modules: Contains third-party libraries.
- public: Contains the index file that dynamic content is injected into, SEO assets, and any static content.
- src: The source code of your React app!
- .gitingore: Tells git to ignore various files and folders, including node_modules. (Note: CRA initializes a git repository).
- package.json: Describes your React app, defines scripts and dependencies.
- README.md: Provides instructions on how to work with React.
- yarn.lock: Locks dependency versions for the local repository.

```
├── node_modules
├── public
|    ├── favicon.ico
|    ├── index.html
|    ├── logo192.png
|    ├── logo512.png
|    ├── manifest.json
|    └── robots.txt
├── src
|    ├── App.css
|    ├── App.js
|    ├── App.test.js
|    ├── index.css
|    ├── index.js
|    ├── logo.svg
|    ├── reportWebVitals.js
|    └── setupTest.js
├── .gitignore
├── package.json
├── README.md
└── yarn.lock
```

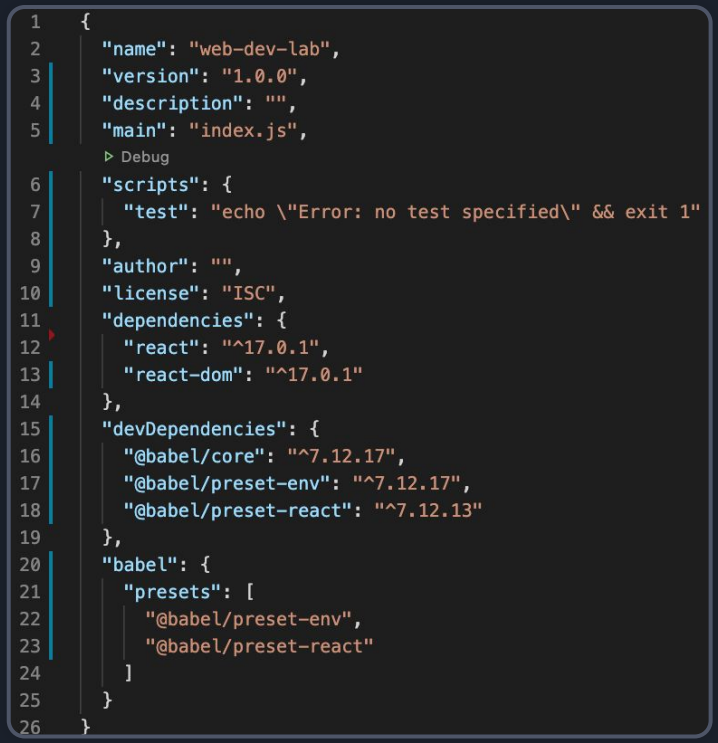
CRA Folder Structure II

- index.js: The entry point for your application.
- App.js: The top-most component of React.
- App.css: (Generally) CSS that applies application-wide.
- reportWebVitals.js: Allows for measuring and analyzing performance of your application.
- setupTests.js, App.test.js: Define tests to be run with npm run test.

```
├── node_modules
├── public
|    ├── favicon.ico
|    ├── index.html
|    ├── logo192.png
|    ├── logo512.png
|    ├── manifest.json
|    └── robots.txt
├── src
|    ├── App.css
|    ├── App.js
|    ├── App.test.js
|    ├── index.css
|    ├── index.js
|    ├── logo.svg
|    ├── reportWebVitals.js
|    └── setupTest.js
├── .gitignore
├── package.json
├── README.md
└── yarn.lock
```

Package.json Structure

- Scripts are terminal commands to be run when npm run scriptName is used.
- Libraries may sometimes override scripts, as in the case of test.
- Dependencies are libraries installed with npm install packageName@version [--save-dev | --save-prod].
- Libraries may add or require additional sections, as in the case of babel.



React Step 2: Modifying the App

- In a *separate* terminal window, start the development server with npm start or yarn start.
- Open a browser window to localhost:3000 if one does not automatically open.
- Open App.js in a code/text editor. VSCode is recommended.
- Change line 10 (Edit <code>src/App.js</code> and save to reload.) to some text of your choice, then save the file.
- Observe the application automatically update in your browser.

React Step 3: Create A Component I

- Steps to creating a component:
- Create a new class which extends the React component base class.
- Define the classes render() method, which is used by React when this class is to be displayed to users.
- Define helper methods as needed (not done here).
- Export the class to make it available for use upon import.
- Import the class and invoke it using its JSX.

React Step 3: Create A Component II

- Create a new folder named List within the src directory.
- Within the List directory, create the following files:
  - List.js
  - ListItem.js
  - ListControls.js
  - List.css

React Step 3: Create A Component III

Add the following to List.js:

```
import React from "react"

class List extends React.Component {
   render() {
       return(
           <div>
               <p>
                   Hello, world!
               </p>
           </div>
       )
   }
}

export default List
```

Note:

Functional components are often preferred over class components. We use a class here for simplicity.

React Step 3: Create A Component IV

And modify App.js:

Note:

<List /> is the JSX way of creating an instance of the List class.

```
import logo from './logo.svg';
import './App.css';
import List from './List/List';

function App() {
 return (
   <div className="App">
     <header className="App-header">
       <img src={logo} className="App-logo" alt="logo" />
       <List />
     </header>
   </div>
 );
}

export default App;
```

React Step 3: Create A Component V

Your app should now look like this:

React Step 4: Working With Props I

- Properties, or props, are like HTML attributes.
- Props are also like function arguments.
- Props pass data from parent elements to child components.
    - “Child component” in this case refers to composition, not Java-like inheritance. E.g., List is a child component of App.
- Components can render differently depending on the value of a prop.

React Step 4: Working With Props II

- Replace “Hello, world! on line 8 of List.js with {this.props.text}
- Replace <List /> on line 10 of App.js with <List text=”This is some text.” />
- How could we change List to conditionally render text in bold or italics?

React Step 4: Working With Props III

- One way is to use the ternary operator.
- Note that JS logic and variables are separated from HTML/JSX elements using brackets, i.e. {}.
- You could also use an if-else block and/or create a helper function outside of render().

```
1   import React from "react"
2
3   class List extends React.Component {
4       render() {
5           return(
6               <div>
7                   <p>
8                       {this.props.text.length % 2 ? (
9                           <b>{this.props.text}</b>
10                      ) : (
11                          <i>{this.props.text}</i>
12                      )}
13                  </p>
14              </div>
15          )
16      }
17  }
18
19  export default List
```

React Step 5: Working With State I

- Components can have state data that is local to them and that they control.
- State can be used to keep track of dynamic information, e.g. the elements of a list.
- Components can use props to update their state.
- Components can update their state in response to user input or actions.
- Components can also update their state in response to other events such as errors, timers, network packets, etc.

React Step 5: Working With State II

- Add a state object to the List class which includes a listItems array.

```
state = {
    listItems: [],
}

render() {
   …

}
```

- Reference listItems state variable using this.state.listItems.
- What might the structure of a list item look like?
- How might we display the list items?

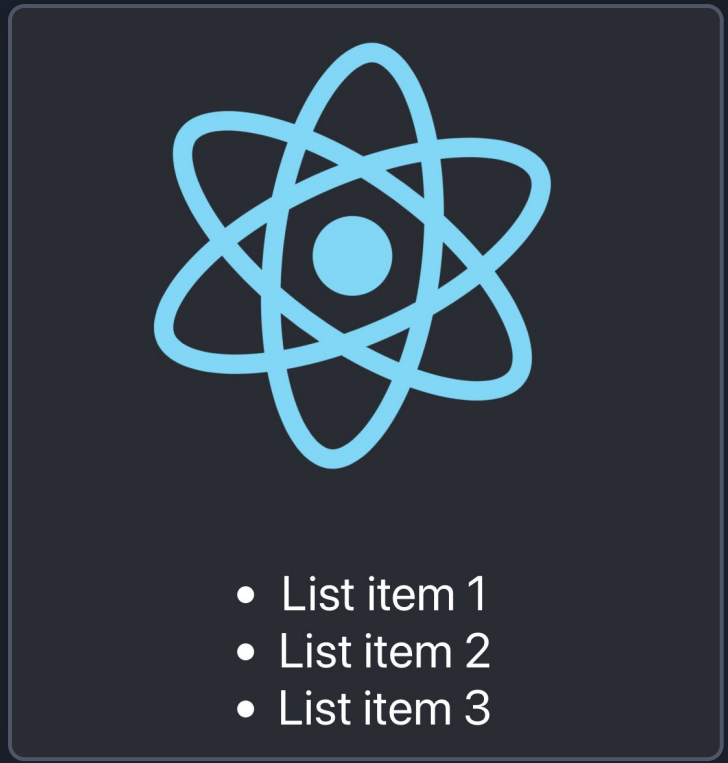
React Step 5: Working With State III

- List items might have *text* and an *ID*.
  - IDs allow us to iterate over the list by ensuring each element has a unique key.
- The Array.map() function can be used to render a list item for each element in itemList.

```
import React from "react"

class List extends React.Component {
    state = {
        listItems: [
            {
                id: 1,
                text: "List item 1"
            },
            {
                id: 2,
                text: "List item 2"
            },
            {
                id: 3,
                text: "List item 3"
            },
        ],
    }

    render() {
        return(
            <ul>
                {this.state.listItems.map(item => (
                    <li>{item.text}</li>
                ))}
            </ul>
        )
    }
}

export default List
```

React Step 5: Working With State IV

Your app should now look like this:



React Step 6: Adding Logic I

- We use this.setState(varName: newValue) to update a component's state.
  - Do not directly modify state; use setState().
  - Be careful not inadvertently modify state when using objects, e.g. arrays. Create a copy of the object, modify as needed, then use setState().
- We can add functions before the render() method to handle various tasks, e.g. adding or removing a list item.
- We can pass function references via props.

React Step 6: Adding Logic II

```
23      addItem = (text) => {
24          var newItem = {id: this.state.listItems.length+1, text: text}
25          var newArray = [...this.state.listItems]
26          newArray.push(newItem)
27          this.setState({listItems: newArray})
28      };
29
30      removeItem = (id) => {
31          var newArray = [...this.state.listItems].filter((item) => item.id !== id)
32          this.setState({listItems: newArray})
33      };
```

React Step 6: Adding Logic III

- As the application gains complexity, we create additional components for features that may be repeated in other areas of our application.

```
35      render() {
36          return(
37              <div>
38                  <ul>
39                      {this.state.listItems.map(item => (
40                          <ListItem
41                              key={item.id}
42                              id={item.id}
43                              text={item.text}
44                              removeItem={this.removeItem}
45                          />
46                      ))}
47                  </ul>
48                  <ListControls addItem={this.addItem} />
49              </div>
50          )
51      }
52  }
```

React Step 6: Adding Logic IV

- We handle events using built-in or custom event handlers, supplying relevant methods from props as needed.

```
1   import React from "react"
2
3   class ListItem extends React.Component {
4       render() {
5           return(
6               <li>
7                   {this.props.text}
8                   <button onClick={() => this.props.removeItem(this.props.id)}>Remove</button>
9               </li>
10          )
11      }
12  }
13
14  export default ListItem
```

React Step 6: Adding Logic IV

- We handle events using built-in or custom event handlers, supplying relevant methods from props as needed.

```
1   import React from "react"
2
3   class ListItem extends React.Component {
4       render() {
5           return(
6               <li>
7                   {this.props.text}
8                   <button onClick={() => this.props.removeItem(this.props.id)}>Remove</button>
9               </li>
10          )
11      }
12  }
13
14  export default ListItem
```

React Step 6: Adding Logic IV

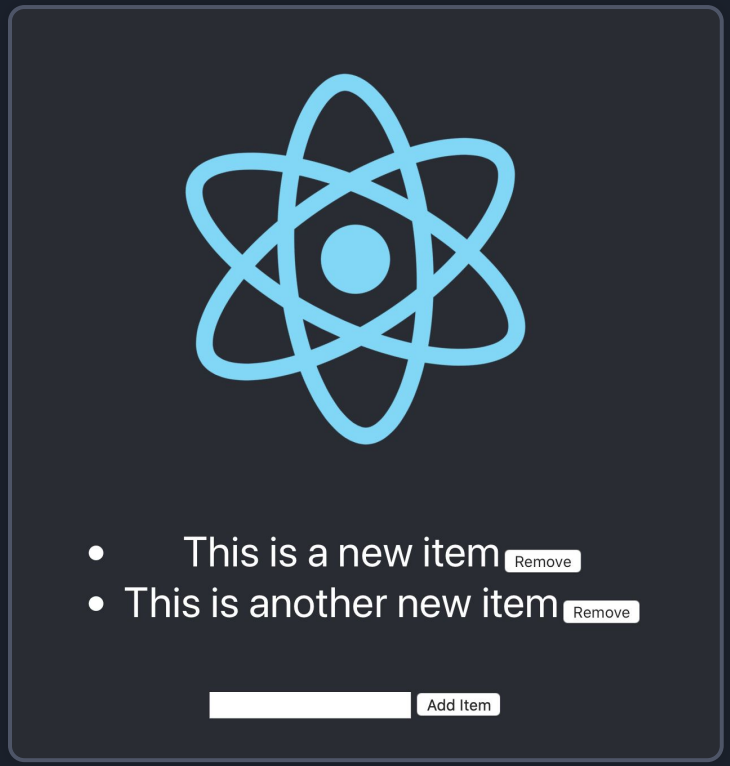
- We can handle input by detecting events (e.g. key press) and updating state to reflect the change.
- We can call methods passed via props to update the state of parent components.

```
import React from "react"

class ListControls extends React.Component {
    state = {
        text: ""
    }

    onChange = e => {
        this.setState({text: e.target.value})
    }

    handleSubmit = e => {
        e.preventDefault();
        this.props.addItem(this.state.text)
        this.setState({
            text: ""
        })
    }

    render() {
        return(
            <form onSubmit={this.handleSubmit}>
                <input type="text" name="text" onChange={this.onChange} />
                <input type="submit" value="Add Item" />
            </form>
        )
    }
}

export default ListControls
```

React Step 6: Adding Logic IV

Your app should now look like this:

You can modify the style of UI elements in List.css or App.css.



Resources I

All code for this lab can be found on GitHub:

https://github.com/SKaplanOfficial/COS420-React-Lab

Feel free to reach out to me with any questions at:

stephen.kaplan@maine.edu

Resources II

- React Website - Tutorials + Documentation - *https://reactjs.org*
- freeCodeCamp React Course (Free) - *https://www.youtube.com/watch?v=DLX62G4lc44*
- Awesome React Resource List - *https://github.com/enaqx/awesome-react*
- Learning React: Functional Web Development with React and Redux
  - Access via *https://go.oreilly.com/university-of-maine-fogler-library-orono*
- Alternative to Create React App - https://dev.to/nikhilkumaran/don-t-use-create-react-app-how-you-can-set-up-your-own-reactjs-boilerplate-43l0

Resources III

- React Todo List Tutorial (From which this tutorial borrows from) - *https://ibaslogic.com/react-tutorial-for-beginners/*
- VSCode - https://code.visualstudio.com
  - Also look into VSCode extensions.
- Airbnb React/JSX Style Guide - https://airbnb.io/javascript/react/
- Beginner's Guide to React JSX - https://dev.to/bipinrajbhar/the-beginner-s-guide-to-react-jsx-leg