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,
  - Applications, which provide various functions such as hosting a website or monitoring traffic, and
  - Resources, such as file storage space.

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 JavaScript 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.
  - Uses JSX, transpiles down to JS/HTML, runs on Node.JS backend by default
- 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

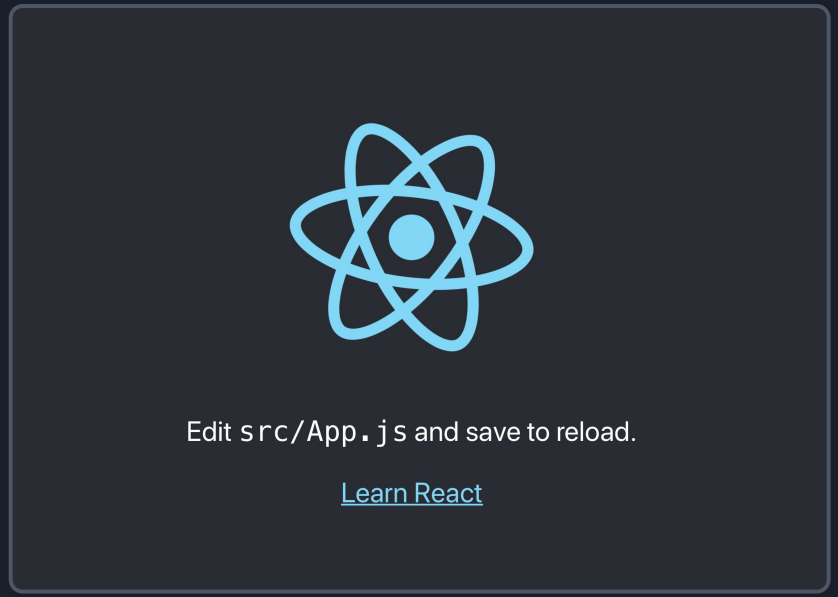
React Step 1: Create React App

- cd into an appropriate folder, then run npx create-react-app react-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 2: Start the Development Server

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



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 or package-lock.json: 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.
- Scripts often reference third-party party libraries.
- Dependencies are libraries installed with npm install packageName@version [--save-dev | --save-prod].
- Additional configuration rules: https://docs.npmjs.com/cli/v8/configuring-npm/package-json

```
{
  "name": "react-lab",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.2",
    "@testing-library/react": "^12.1.2",
    "@testing-library/user-event": "^13.5.0",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "5.0.0",
    "web-vitals": "^2.1.4"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

React Step 3: 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 Components

- A component represents a feature, functionality, UI element, or some other information.
- Components are small and single-purpose, e.g.: Navigation Menu, Sidebar, Contact Form, Button, TextField, SwipeHandler
- Components are composed of several subcomponents, e.g.: App < NavigationMenu < NavigationElement < Button, Link < Text
    - Raw HTML elements are also child elements of components.
- You define the size and scope of your components.
- Can you think of other examples of components that a website might use?

React Step 4: Create A Component I

- Steps to creating a component:
  - Create a new function which returns a single JSX element.
    - The JSX element can any number of child elements.
  - Define helper methods as needed.
  - Export the functional component to make it available for use upon import.
  - Import the component and invoke it using its JSX.

React Step 4: 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 4: Create A Component III

Add the following to List.js:

```
function List() {
    return(
        <div>
            <p>
                Hello, world!
            </p>
        </div>
    );
}

export default List;
```

React Step 4: 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 4: Create A Component V

Your app should now look like this:

React Step 5: 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 5: Working With Props II

- Replace () on line 1 of List.js with (props)
- Replace Hello, world! on line 8 of List.js with {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 5: 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, create a helper function, or add logic before return().

```
function List(props) {
    return (
        <div>
            <p>
                {props.text.length % 2 ? (
                    <b>{props.text}</b>
                ) : (
                    <i>{props.text}</i>
                )}
            </p>
        </div>
    );
}

export default List;
```

React Step 6: Working With State I

- Components can have state variables that persist as long as the component is active.
- 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 6: Working With State II

- Add a state hook to the List component which includes a listItems array.

```
const [listItems, setListItems] = React.useState([]);
```

- You will also need to import the React library at the top of the List.js file:

```
import React from "react";
```

- Reference listItems state variable using {listItems}, update it by calling setListItems()
- How would we structure a list item? How might we display the list items?

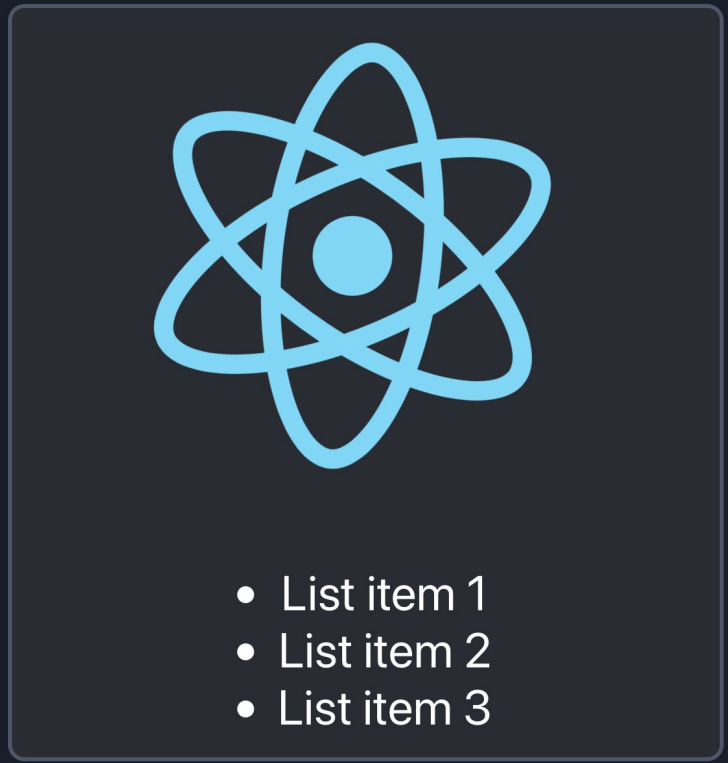
React Step 6: Working With State III

- 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() method can be used to render a list item for each element in itemList.

```
import React from "react";

function List(props) {
    const [listItems, setListItems] = React.useState([
        {
            id: 1,
            text: "List item 1"
        },
        {
            id: 2,
            text: "List item 2"
        },
        {
            id: 3,
            text: "List item 3"
        },
    ]);

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

export default List;
```

React Step 6: Working With State IV

Your app should now look like this:



React Step 7: Adding Logic I

- React.useState() returns references to a state variable and an update method.
  - Do not directly modify state; use the setState() method.
  - 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 JS code to handle various tasks, e.g. adding or removing a list item. What might this look like?
- We can also pass function references via props.

React Step 7: Adding Logic II

```
20      const addItem = (text) => {
21          var newItem = {id: listItems.length+1, text: text}
22          var newArray = [...listItems]
23          newArray.push(newItem)
24          setListItems(newArray)
25      };
26
27      const removeItem = (id) => {
28          var newArray = [...listItems].filter((item) => item.id !== id)
29          setListItems(newArray)
30      };
```

React Step 7: Adding Logic III

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

```
34      return (
35          <div>
36              <ul>
37                  {listItems.map(item => (
38                      <ListItem
39                          key = {item.id}
40                          id = {item.id}
41                          text = {item.text}
42                          removeItem = {removeItem}
43                      />
44                  ))}
45              </ul>
46              <ListControl addItem={addItem} />
47          </div>
48      );
```

React Step 7: 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   function ListItem(props) {
4       return (
5           <li>
6               {props.text}
7               <button onClick={() => props.removeItem(props.id)}>Remove</button>
8           </li>
9       );
10  }
```

React Step 7: Adding Logic V

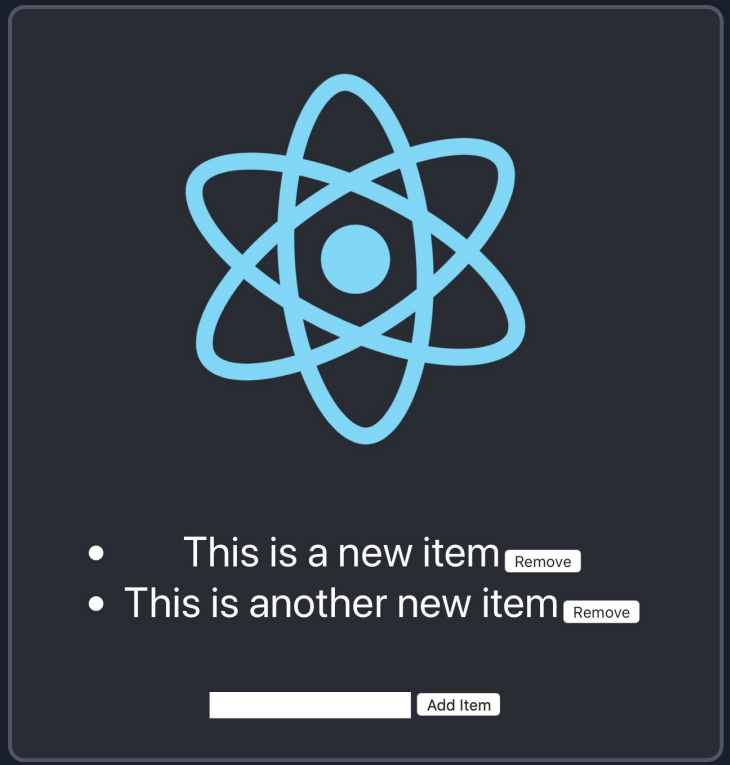
- We can handle input by detecting events (e.g. key press) and updating state to reflect the change.
- We can call methods passed via props to update the state of parent components.

```
1   import React from "react";
2
3   function ListControl(props) {
4       const [text, setText] = React.useState("");
5
6       const onChange = e => {
7           setText(e.target.value);
8       }
9
10      const handleSubmit = e => {
11          e.preventDefault();
12          props.addItem(text)
13          setText("");
14      }
15
16      return (
17          <form onSubmit={handleSubmit}>
18              <input type="text" name="text" value={text} onChange={onChange} />
19              <input type="submit" value="Add item" />
20          </form>
21      );
22  }
23
24  export default ListControl;
```

React Step 7: Adding Logic VI

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
- Guide to Functional Components - https://www.twilio.com/blog/react-choose-functional-components