Exercise 5: Creating Single Page Applications with React.js

1 Introduction

In the previous exercise, we introduced ***JavaScript*** as the language for programming the browser, and ***jQuery*** as a popular JavaScript library that simplified common tasks such as binding to the DOM and interacting with the user. In this exercise, we build on the JavaScript skills to introduce ***React.js***, a popular JavaScript library for building ***Single Page Applications*** (***SPAs***).

2 Exercises

This section presents ***React.js*** examples to program the browser, interact with the user, and generate dynamic HTML. After you work through the examples you will apply the skills while creating a clone of ***Tuiter*** on your own. Open the ***tuiter-react-web-app*** project you created in assignment 0. Open the ***tuiter-react-web-app*** directory, and then the ***src*** directory. Do all your work under the ***src*** directory of your project.

2.1 Implementing Single Page Applications

Single Page Applications (***SPAs***) render all their content dynamically into a single HTML document including navigation between various screens, without actually navigating away from the original HTML document. ***React.js*** achieves this by declaring a single HTML element where all the content is rendered by the ***ReactDOM*** library into a DIV with a ***root*** ID in the ***public/index.html*** document. Make sure ***public/index.html*** contains the ***div #root*** as shown below.

public/index.html

```
<body>
 <div id="root"></div>
</body>
```

The ***React.js*** application is implemented in ***src/index.js*** as shown below. The code imports the ***React*** and ***ReactDOM*** libraries.

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 <React.StrictMode>
```

```
    <App />
  </React.StrictMode>
);
```

ReactDOM uses ***document.getElementById('root')*** to retrieve a reference to the ***DOM*** element declared in ***index.html***. ***ReactDOM*** then creates an instance of ***App*** and appends its output to the element whose ID is ***root***. The ***src/App.js*** is the entry point of the ***React.js*** application we're going to build and it might contain code generated by the ***create-react-app*** tool we used to create the project at the beginning of the course. Let's replace the content of ***src/App.js*** with the code below. It's basically a function called ***App*** that returns an ***H1*** element greeting the world. Note how the return statement is returning an ***HTML tag***, not an ***HTML string***. This is possible because ***React.js*** uses a library called ***JSX*** or ***JavaScript XML***. ***JSX*** allows mixing and matching ***JavaScript*** and ***XML*** seamlessly and ***HTML*** is just a particular flavor of ***XML***. This syntax greatly simplifies integrating HTML and JavaScript as if they were two sides of the same coin.

src/App.js

```
function App() {
    return (
        <h1>Hello World!</h1>
    );
}
export default App;
```

To test, start the React application using ***npm*** as shown below. Run the command from the root directory of your project.

```
npm  start
```

Confirm the browser refreshes with the ***Hello World!***.

2.2 Installing CSS libraries Bootstrap and Bootstrap Icons

We're going to keep using the same styling libraries we've been using so far: Bootstrap and Fontawesome. We could use the same bootstrap CSS library we've been using in previous assignments, but we are going to install it as a React library. Install Bootstrap from the root of the project as follows

```
npm install bootstrap
```

Let's also install Bootstrap's icon library. This is an alternative to the Fontawesome icon library. Feel free to use either set of icons.

```
npm install bootstrap-icons
```

Once the libraries are installed you can load them by importing them from the ***src/index.js*** as shown below. Confirm that the browser refreshes with Bootstrap styling.

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import reportWebVitals from './reportWebVitals';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap-icons/font/bootstrap-icons.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 <React.StrictMode>
   <App />
 </React.StrictMode>
);
```

2.3 Implementing the Exercises component

Let's create a folder ***src/Exercises*** and work on all the exercises in ***src/Exercises/index.js***. Add the following content in the new ***index.js*** file and import the new component in the ***App.js***.

src/Exercises/index.js

```
function Exercises() {
   return(
      <div>
         <h1>Exercise 5</h1>
      </div>
   );
}
export default Exercises;
```

In ***App.js***, import the ***Exercises*** component as shown below. Wrap the HTML content in a DIV element with the ***container*** class. Note that in React.js we use ***className*** instead of ***class***. Confirm the application renders as shown below.

App.js

```
import Exercises from "./Exercises";
function App() {
   return (
      <div className="container">
         <h1>Hello World!</h1>
         <Exercises/>
      </div>
   );
}
export default App;
```

Hello World!

Exercise 5

2.4 Breaking out exercises into separate components

The Exercises component will hold all the lab exercises for this assignment as well as future assignments. Let's break out each exercise into its own separate component. In a new file in ***src/Exercises/e5/index.js***, create the following component.

src/Exercises/e5/index.js

```
function Exercise5() {
 return (
   <div>
     <h1>Exercise 5</h1>
   </div>
 );
}
export default Exercise5;
```

Then import the new component into the ***Exercises*** component. Confirm the application renders as before. In later Exercises you'll be creating separate components, one for each Exercise, that contain the exercises for that specific Exercise. You'll import them into the ***Exercises*** component so they are all accessible in one place.

Exercises/index.js

```
import Exercise5 from "./e5";

function Exercises() {
 return (
   <div>
     <h1>Exercise 5</h1>
     <Exercise5/>
   </div>
 );
}
export default Exercises;
```

Hello World!

Exercise 5

2.5 Breaking out Hello World into a separate component

One of React.js strengths is that it encourages breaking up large applications into smaller parts or ***components*** you can then assemble into sophisticated user interfaces. Let's create another React.js component by breaking out the ***Hello World*** H1 element into a separate JavaScript file as shown below. In ***src/Exercises/e5/hello-world.js*** create a ***HelloWorld*** component as shown below.

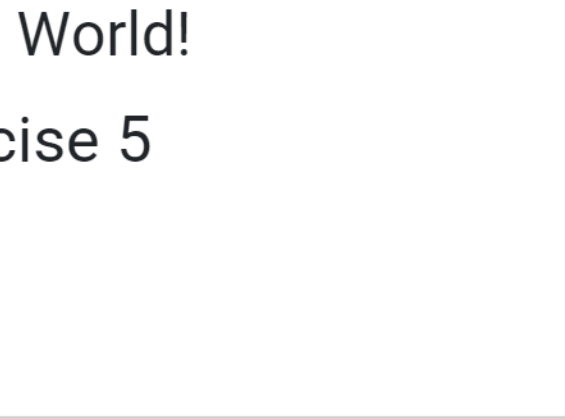
src/Exercises/e5/hello-world.js

```
function HelloWorld() {
   return(
       <h1>Hello World!</h1>
   );
};
export default HelloWorld;
```

We can then import the new component in ***src/App.js*** as shown below. Note the missing ***.js*** optional file extension in the ***HelloWorld*** import statement. Also note the new ***<HelloWorld/>*** tag matching the name of the import, file name, and function name.

App.js

```
import Exercises from "./Exercises";
import HelloWorld from "./Exercises/e5/hello-
world.js";

function App() {
   return (
      <div className="container">
         <HelloWorld/>
         <Exercises/>
      </div>
   );
}
export default App;
```



2.6 Creating a Tuiter placeholder component

Let's create another component we'll use later to implement the ***Tuiter*** application. Let's create the component in ***src/tuiter/index.js*** with the content below. This will be a placeholder for your assignment.

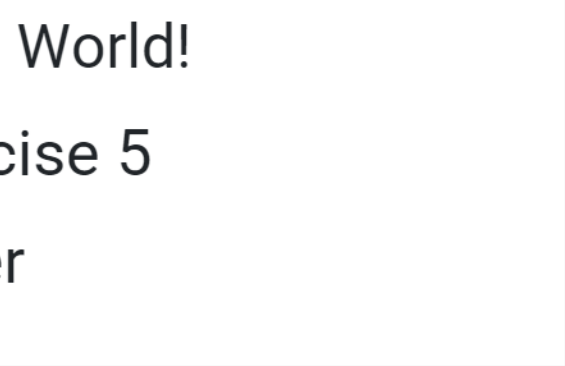
src/tuiter/index.js

```
function Tuiter() {
   return(
      <div>
         <h1>Tuiter</h1>
      </div>
   );
}
export default Tuiter
```

Import the new Tuiter component in ***App.js*** as shown below. Confirm the output renders as shown below.

App.js

```
import Exercises from "./Exercises";
import HelloWorld from "./Exercises/e5/hello-
world.js";
import Tuiter from "./tuiter";

function App() {
   return (
      <div className="container">
         <HelloWorld/>
         <Exercises/>
         <Tuiter/>
```



```
      </div>
   );
}
export default App;
```

2.7 Implementing navigation in Single Page Applications

Earlier we mentioned that ***Single Page Applications*** (***SPA***s) implement applications by dynamically rendering all content into a single HTML document and that we rarely or never navigate away from that one HTML document, so you might ask, how do we break up a large website or application into several screens? The answer is that React.js can accomplish the same functionality by swapping different screens in and out of the single HTML document giving the illusion of navigating between multiple screens. Instead of building this feature ourselves from scratch, we'll use a popular navigation library called React Router. To practice navigating between various screens, let's implement navigation between the components we've created so far: ***HelloWorld***, ***Exercises***, and ***Tuiter***. To implement navigation, we'll need to install the ***React Router*** library from the command line as shown below. Run the command from the root of the project.

```
npm install react-router
```

The React Router library can be used to implement navigation in all kinds of devices including Web applications, mobile, and desktop. To implement navigation in Web application, also install the ***React Router DOM*** library as follows:

```
npm install react-router-dom
```

Once the library has fully downloaded and installed, let's use the ***BrowserRouter*** to implement navigation as shown below. The ***BrowserRouter*** tag sets up the base mechanism to navigate between multiple components. In this case we're going to navigate between the three components within the ***BrowserRouter*** tag, e.g., ***HelloWorld***, ***Exercises*** and ***Tuiter***.

App.js

```
import Exercises from "./Exercises";
import HelloWorld from "./Exercises/e5/hello-world.js";
import Tuiter from "./tuiter";

import {BrowserRouter} from "react-router-dom";

function App() {
   return (
       <BrowserRouter>
           <div className="container">
               <HelloWorld/>
               <Exercises/>
               <Tuiter/>
           </div>
       </BrowserRouter>
   );
```

```
}
export default App;
```

To navigate between components, we use the ***Route*** component from ***React Router*** to declare ***paths*** and map them to corresponding component we want to render for that ***path***. Update your code as shown below.

App.js

```
import Exercises from "./Exercises";
import HelloWorld from "./Exercises/e5/hello-
world.js";
import Tuiter from "./tuiter";
import {BrowserRouter} from "react-router-dom";
import {Routes, Route} from "react-router";

function App() {
  return (
    <BrowserRouter>
      <div className="container">
        <Routes>
          <Route path="/Exercises"
                 element={<Exercises/>}/>
          <Route path="/hello"
                 element={<HelloWorld/>}/>
          <Route path="/tuiter"
                 element={<Tuiter/>}/>
        </Routes>
      </div>
    </BrowserRouter>
  );
}
export default App;
```

http://localhost:3000/hello

Hello World!

http://localhost:3000/Exercises

Exercise 6

http://localhost:3000/tuiter

Tuiter

Having declared the routes, now the components won't all render at the same time in the same screen. Instead, they will render when the URL in the browser matches the path declared in their parent Route. To test this, refresh your browser and navigate to http://localhost:3000/hello and confirm the ***Hellow World!*** message appears. Then confirm navigating to http://localhost:3000/Exercises displays ***Exercise 5***. Then confirm navigating to http://localhost:3000/tuiter displays ***Tuiter***.

We can declare the ***Exercise*** component as the default landing screen by declaring it the ***index*** and removing its ***path*** attribute as shown below. Refresh the browser and confirm that the current Exercise component is now the default screen.

App.js

```
<BrowserRouter>
   <div className="container">
       <Routes>
           <Route index path="/Exercises"
                  element={<Exercises/>}/>
           <Route path="/hello"
                  element={<HelloWorld/>}/>
           <Route path="/tuiter"
```

http://localhost:3000/

Exercise 6

```
                element={<Tuiter/>}/>
        </Routes>
    </div>
</BrowserRouter>
```

2.8 Navigating with links in SPAs

Instead of typing the links in a browser's navigation bar, we can create hyperlinks in our components that navigate between them. The examples below implement navigation between all three components created so far. Refresh the browser and confirm you can navigate between all components.

e5/index.js

```
import {Link}
  from "react-router-dom";

function Exercise5() {
 return(
  <div>
   <Link to="/">Exercise</Link> |
   <Link to="/hello">Hello</Link> |
   <Link to="/tuiter">Tuiter</Link>
   <h1>Exercise 6</h1>
  </div>
 )
}
export default Exercise5;
```

Exercise |Hello |Tuiter

Exercise 5

e5/hello-world.js

```
import {Link}
  from "react-router-dom";

function HelloWorld() {
 return(
  <div>
   <Link to="/"> Exercise </Link> |
   <Link to="/hello">Hello</Link> |
   <Link to="/tuiter">Tuiter</Link>
   <h1>Hello World!</h1>
  </div>
 )
};
export default HelloWorld;
```

Exercise |Hello |Tuiter

Hello World!

tuiter/index.js

```
import {Link}
  from "react-router-dom";

function Tuiter() {
 return(
  <div>
   <Link to="/"> Exercise </Link> |
   <Link to="/hello">Hello</Link> |
   <Link to="/tuiter">Tuiter</Link>
   <h1>Tuiter</h1>
  </div>
 )
}
export default Tuiter
```

Exercise |Hello |Tuiter

Tuiter

2.9 Implementing a Navigation component

The navigation links in the three components, ***Exercises***, ***HelloWorld***, and ***Tuiter***, would be best implemented as a reusable component as shown below.

src/nav.js

```
import {Link} from "react-router-dom";

function Nav() {
 return (
   <div>
     <Link to="/">Exercise</Link> |
     <Link to="/hello">Hello</Link> |
     <Link to="/tuiter">Tuiter</Link>
   </div>
 )
}

export default Nav;
```

The component can then be imported into the ***HelloWorld***, ***Exercises***, and ***Tuiter*** component as shown below. Reload your application and confirm the navigation still works.

Exercises/index.js

```
import Exercise5 from "./e5";
import Nav from "../nav";
function Exercises() {
 return (
   <div>
     <Nav/>
     <Exercise5/>
   </div>
 );
}
export default Exercises;
```

Exercise |Hello |Tuiter

Exercise 5

Exercises/e5/hello-world.js

```
import Nav from "../nav";
function HelloWorld() {
 return (
   <div>
     <Nav/>
     <h1>Hello World!</h1>
   </div>
 );
}
export default HelloWorld;
```

Exercise |Hello |Tuiter

Hello World!

tuiter/index.js

```
import Nav from "../nav";
function Tuiter() {
 return (
   <div>
     <Nav/>
     <h1>Tuiter</h1>
   </div>
 )
}
export default Tuiter
```

Exercise |Hello |Tuiter

Tuiter

2.10 Working with HTML classes

Let's start practicing simple things, like classes and styles. Under the ***e5*** folder, create another folder called ***classes*** and create the following component and styling files.

e5/classes/index.js

```
import './index.css';

function Classes() {
 return (
   <div>
     <h2>Classes</h2>
     <div className="wd-bg-yellow wd-fg-black wd-padding-10px">
       Yellow background
     </div>
     <div className="wd-bg-blue wd-fg-black wd-padding-10px">
       Blue background
     </div>
     <div className="wd-bg-red wd-fg-black wd-padding-10px">
       Red background
     </div>
   </div>
 )
};
export default Classes;
```

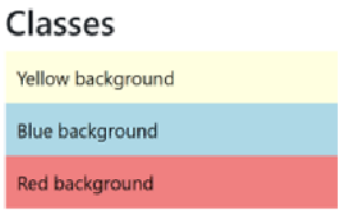
e5/classes/index.css

```
.wd-bg-yellow {
 background-color: lightyellow;
}

.wd-bg-blue {
 background-color: lightblue;
}

.wd-bg-red {
 background-color: lightcoral;
}

.wd-fg-black {
 color: black;
}

.wd-padding-10px {
 padding: 10px
}
```

From the ***exercise5*** component, import the new ***Classes*** component as shown below. Confirm the new ***classes*** component renders in the screen as expected.

e5/index.js

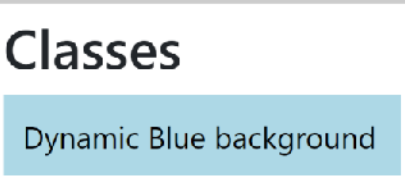
```
import Classes from "./classes";

function Exercise5() {
 return (
   <div>
     <h1>Exercise 5</h1>
     <Classes/>
   </div>
 );
}
export default Exercise5;
```



The previous example used static classes such as ***wd-bg-yellow***. Instead, we could calculate the class we want to apply based on any convoluted logic. Here's an example of creating the classes dynamically by concatenating a ***color*** constant. Refresh the screen and confirm components render as expected.

e5/classes/index.js

```
function Classes() {
 const color = 'blue';
 return (
   <div>
     <h2>Classes</h2>
     <div className={`wd-bg-${color} wd-fg-black wd-padding-10px`}>
       Dynamic Blue background</div>
```



Even more interesting is using expressions to conditionally choose between a set of classes. The example below uses either a ***red*** or ***green*** background based on the ***dangerous*** constant. Try with ***dangerous true*** and ***false*** and confirm it renders red or green as expected.

e5/classes/index.js

```
function Classes() {
 const color = 'blue';
 const dangerous = true;
 return (
   <div>
     <h2>Classes</h2>
     <div className={`${dangerous ? 'wd-bg-red' : 'wd-bg-green'}
                                  wd-fg-black wd-padding-10px`}>
       Dangerous background</div>
```

2.11 Working with the HTML style attribute

In HTML the ***styles*** attribute accepts a CSS string to style the element applied to. In React.js, the ***styles*** attribute does not accept a string; instead, it accepts a JSON object where the properties are CSS properties and the values are CSS values. To practice how this works, implement the ***Styles*** component below in ***e5/styles*** and then import it into the ***Exercise5*** component as shown below. The ***styles*** component (***styles/index.js***) declares constant JSON objects that can be applied to elements using the ***styles*** attribute. Alternatively, the styles attribute accepts a JSON literal object instance which results in a weird syntax of double curly brackets as shown below. Also note that the ***Styles*** component is implemented using the new arrow function syntax. Refresh the browser and confirm the browser renders as expected. Note we use ***background-color*** instead of ***backgroundColor***.

e5/styles/index.js

```
const Styles = () => {
 const colorBlack = {
   color: "black"
 }
 const padding10px = {
   padding: "10px"
 }
 const bgBlue = {
   "backgroundColor": "lightblue",
   "color": "black",
   ...padding10px
 };
 const bgRed = {
   "backgroundColor": "lightcoral",
   ...colorBlack,
   ...padding10px
 };
 return(
   <div>
     <h1>Styles</h1>
     <div style={{"backgroundColor": "lightyellow",
       "color": "black", padding: "10px"}}>
       Yellow background</div>
     <div style={bgRed}>
       Red background</div>
     <div style={bgBlue}>
       Blue background</div>
   </div>
 );
};
export default Styles;
```

e5/index.js

```
import Classes from "./classes";
import Styles from "./styles";

function Exercise5 () {
 return (
   <div>
     <h1> Exercise 5</h1>
     <Styles/>
     <Classes/>
   </div>
 )
}
export default Exercise5;
```

2.12 Generating conditional output

Ok, enough styling. Let's play around with rendering content based on some logic. The following example decides to render one content versus another based on a simple boolean constant ***loggedIn***. If the user is ***loggedIn***, then the component renders a greeting, otherwise suggests the user should login. Implement the example in ***e5/conditional-output/conditional-output-if-else.js*** with the following code.

e5/conditional-output/conditional-output-if-else.js

```
const ConditionalOutputIfElse = () => {
 const loggedIn = true;
 if(loggedIn) {
   return (<h2>Welcome If Else</h2>);
 } else {
   return (<h2>Please login If Else</h2>);
 }
};
export default ConditionalOutputIfElse;
```

A more compact way to achieve the same thing is to include the conditional content in a boolean expression that short circuits the content if its false or evaluates the expression if it's true. Implement the equivalent component below in ***e5/conditional-output/conditional-output-inline.js***.

e5/conditional-output/conditional-output-inline.js

```
const ConditionalOutputInline = () => {
 const loggedIn = false;
 return (
   <>
     { loggedIn && <h2>Welcome Inline</h2>}
     {!loggedIn && <h2>Please login Inline</h2>}
   </>
 );
};
export default ConditionalOutputInline;
```

Merge both components into a single component as shown below and then import the new component into the ***e5/index.js***. Confirm all components render as expected.

e5/conditional-output/index.js

```
import React from "react";
import ConditionalOutputIfElse
  from "./conditional-output-if-else";
import ConditionalOutputInline
  from "./conditional-output-inline";
const ConditionalOutput = () => {
 return(
   <>
     <ConditionalOutputIfElse/>
     <ConditionalOutputInline/>
   </>
 );
};
export default ConditionalOutput;
```

e5/index.js

```
import Classes from "./classes";
import Styles from "./styles";
import ConditionalOutput
  from "./conditional-output";

function Exercise5() {
 return (
   <div>
     <h1> Exercise 5</h1>
     <ConditionalOutput/>
     <Styles/>
     <Classes/>
   </div>
 );
}
export default Exercise5;
```

2.13 Implementing ToDo List using React

In a previous assignment we created a ***Todo*** list application that rendered a list of todos dynamically using ***JavaScript*** and ***jQuery***. In this section we're going to re-implement the Todo application using React.js. In a new directory ***e5/todo***, implement the ***TodoItem*** complement in a ***todo-item.js*** file as shown below. Import the component into the ***`Exercise5`*** component and confirm that it renders as shown.

e5/todo/todo-item.js

```
const TodoItem = (
 {
   todo = {
     done: true,
     title: 'Buy milk',
     status: 'COMPLETED'
   }
 }) => {
 return (
   <li>
     <input type="checkbox"
            defaultChecked={todo.done}/>
     {todo.title}
     ({todo.status})
   </li>
 );
}
export default TodoItem;
```

Store todo objects in a JSON file as shown below

e5/todo/todos.json

```
[
 { "title": "Buy milk",          "status": "CANCELED",     "done": true   },
 { "title": "Pickup the kids",   "status": "IN PROGRESS",  "done": false  },
 { "title": "Walk the dog",      "status": "DEFERRED",     "done": false  }
]
```

Now let's implement the ***todo-list.js*** as shown below. Import ***todo-list.js*** in ***e5/index.js***, refresh the browser, and confirm the ***TodoList*** renders a list of checkboxes and todo items.

e5/todo/todo-list.js

```
import TodoItem from "./todo-item";
import todos from "./todos.json";
const TodoList = () => {
 return(
   <>
     <h3>Todo List</h3>
     <ul>
       {
         todos.map(todo => {
           return(<TodoItem todo={todo}/>);
         })
       }
     </ul>
   </>
 );
}
export default TodoList;
```