Creating Single Page Applications with React.js

# 1 Introduction

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

# 2 Labs

This section presents ***React.js*** examples to program the browser, interact with the user, and generate dynamic HTML. Use the same project you worked on last assignment. After you work through the examples you will apply the skills while creating a clone of ***Tuiter*** on your own. Using ***IntelliJ***, open the project you created in the previous assignment. From within IntelliJ, use ***File***, ***Open Project***, and navigate to the project directory, (***tuiter-react-web-app***), and click ***Open*** or ***OK***. From within IntelliJ, on the ***Project*** tab, 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. Remove all other content in the body tag from previous assignments.

| ***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!***. Alternatively create a ***Run Configuration*** by selecting ***Edit Configurations*** from the ***Run*** menu. In the ***Run/Debug Configuration*** dialog, click the plus sign and select ***npm***. In the ***Name*** field give a name such as ***run***. From the Scripts dropdown select start. Click ***Ok*** and confirm the configuration appears in the dropdown at the top right of the IDE, next to a green triangular play button. To run the React application select the run configuration and click on the green play button.

## 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 Labs component

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

| ***src/labs/index.js*** |  |
| --- | --- |
| **function** *Labs*() {  **return**(  <**div**>  <**h1**>Assignment 6</**h1**>  </**div**>  );  }  **export default** *Labs*; | |

In ***App.js***, import the ***Labs*** 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** *Labs* **from "./labs"**;  **function** *App*() {  **return** (  <**div className="container"**>  <**h1**>Hello World!</**h1**>  <**Labs**/>  </**div**>  );  }  **export default** *App*; | Hello World!Assignment 6 |

## 2.4 Breaking out assignments into separate components

The Labs component will hold all the lab exercises for this assignment as well as future assignments. Let's break out each assignment into its own separate component. In a new file in ***src/labs/a6/index.js***, create the following component.

| ***src/labs/a6/index.js*** | | |
| --- | --- | --- |
| **function** *Assignment6*() {  **return** (  <**div**>  <**h1**>Assignment 6</**h1**>  </**div**>  );  }  **export default** *Assignment6*; | | |

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

| ***labs/index.js*** | |  |
| --- | --- | --- |
| **import** *Assignment6* **from "./a6"**;  **function** *Labs*() {  **return** (  <**div**>  ~~<~~**~~h1~~**~~>Assignment 6</~~**~~h1~~**~~>~~  <**Assignment6**/>  </**div**>  );  }  **export default** *Labs*; | | Hello World!Assignment 6 |

## 2.5 Breaking out Hello World into a separate component

One of React.js strengths is that it incourages 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/labs/a6/hello-world.js*** create a ***HelloWorld*** component as shown below.

| ***src/labs/a6/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** *Labs* **from "./labs"**;  **import** *HelloWorld* **from "./labs/a6/hello-world"**;  **function** *App*() {  **return** (  <**div className="container"**>  <**HelloWorld**/>  <**Labs**/>  </**div**>  );  }  **export default** *App*; | Hello World!Assignment 6 |

## 2.6 Creating a Tuiter placeholder component

Let's create a nother 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 a later section.

| ***src/tuiter/index.js*** |  |
| --- | --- |
| **function** *Tuiter*() {  **return**(  <**div**>  <**h1**>Tuiter</**h1**>  </**div**>  );  }  **export default** *Tuiter* | |

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

| ***App.js*** |  |
| --- | --- |
| **import** *Labs* **from "./labs"**;  **import** *HelloWorld* **from "./labs/a6/hello-world"**;  **import** *Tuiter* **from "./tuiter"**;  **function** *App*() {  **return** (  <**div className="container"**>  <**HelloWorld**/>  <**Labs**/>  <**Tuiter**/>  </**div**>  );  }  **export default** *App*; | Hello World!Assignment 6Tuiter |

## 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](https://reactrouter.com/web/guides/quick-start). To practice navigating between various screens, let's implement navigation between the components we've created so far: ***HelloWorld***, ***Labs***, 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***, ***Labs*** and ***Tuiter***.

| ***App.js*** |  |
| --- | --- |
| **import** *Labs* **from "./labs"**;  **import** *HelloWorld* **from "./labs/a6/hello-world"**;  **import** *Tuiter* **from "./tuiter"**;  **import** {BrowserRouter} **from "react-router-dom"**;  **function** *App*() {  **return** (  <**BrowserRouter**>  <**div className="container"**>  <**HelloWorld**/>  <**Labs**/>  <**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** *Labs* **from "./labs"**;  **import** *HelloWorld* **from "./labs/a6/hello-world"**;  **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="/labs"**  **element=**{<**Labs**/>}/>  <**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/labs/a6> Assignment 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](http://localhost:3000/a6/hello) and confirm the ***Hellow World!*** message appears. Then confirm navigating to <http://localhost:3000/labs> displays ***Assignment 6***. Then confirm navigating to [http://localhost:3000/tuiter](http://localhost:3000/a6/build) displays ***Tuiter***.

We can declare the ***Lab*** 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 assignment component is now the default screen.

| ***App.js*** |  |
| --- | --- |
| <**BrowserRouter**>  <**div className="container"**>  <**Routes**>  <**Route index ~~path="/labs"~~**  **element=**{<**Labs**/>}/>  <**Route path="/hello"**  **element=**{<**HelloWorld**/>}/>  <**Route path="/tuiter"**  **element=**{<**Tuiter**/>}/>  </**Routes**>  </**div**>  </**BrowserRouter**> | <http://localhost:3000/> Assignment 6 |

## 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.

| ***a6/index.js*** | ***a6/hello-world.js*** | ***tuiter/index.js*** |
| --- | --- | --- |
| **import** {Link}  **from "react-router-dom"**;  **function** Assignment6() {  **return**(  <**div**>  <**Link to="/"**>Lab</**Link**> |  <**Link to="/hello"**>Hello</**Link**> |  <**Link to="/tuiter"**>Tuiter</**Link**>  <**h1**>Assignment 6</**h1**>  </**div**>  )  }  **export default** Assignment6; | **import** {Link}  **from "react-router-dom"**;  **function** HelloWorld() {  **return**(  <**div**>  <**Link to="/"**>Lab</**Link**> |  <**Link to="/hello"**>Hello</**Link**> |  <**Link to="/tuiter"**>Tuiter</**Link**>  <**h1**>Hello World!</**h1**>  </**div**>  )  };  **export default** HelloWorld; | **import** {Link}  **from "react-router-dom"**;  **function** Tuiter() {  **return**(  <**div**>  <**Link to="/"**>Lab</**Link**> |  <**Link to="/hello"**>Hello</**Link**> |  <**Link to="/tuiter"**>Tuiter</**Link**>  <**h1**>Tuiter</**h1**>  </**div**>  )  }  **export default** Tuiter |
| [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Assignment 6 | [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Hello World! | [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Tuiter |

## 2.9 Implementing a Navigation component

The navigation links in the three components, ***Labs***, ***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="/"**>Labs</**Link**> |  <**Link to="/hello"**>Hello</**Link**> |  <**Link to="/tuiter"**>Tuiter</**Link**>  </**div**>  )  }  **export default** *Nav*; | |

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

| ***labs/index.js*** | ***labs/a6/hello-world.js*** | ***tuiter/index.js*** |
| --- | --- | --- |
| **import** *Assignment6* **from "./a6"**;  **import** *Nav* **from "../nav"**;  **function** *Labs*() {  **return** (  <**div**>  <**Nav**/>  <**Assignment6**/>  </**div**>  );  }  **export default** *Labs*; | **import** *Nav* **from "../nav"**;  **function** *HelloWorld*() {  **return** (  <**div**>  <**Nav**/>  <**h1**>Hello World!</**h1**>  </**div**>  );  }  **export default** *HelloWorld*; | **import** *Nav* **from "../nav"**;  **function** *Tuiter*() {  **return** (  <**div**>  <**Nav**/>  <**h1**>Tuiter</**h1**>  </**div**>  )  }  **export default** *Tuiter* |
| [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Assignment 6 | [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Hello World! | [Lab](http://localhost:3000/) |[Hello](http://localhost:3000/hello) |[Tuiter](http://localhost:3000/tuiter) Tuiter |

## 2.10 Working with HTML classes

Let's start practicing simple things, like classes and styles. Under the ***labs/a6*** folder, create another folder called ***classes*** and create the following component and styling files.

| ***index.js*** | ***index.css*** |
| --- | --- |
| **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*; | .**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**: 10**px**  } |

From the ***Assignment6*** component, import the new ***Classes*** component as shown below. Confirm the new ***classes*** component renders in the screen as expected.

| ***labs/a6/index.js*** |  |
| --- | --- |
| **import** *Classes* **from "./classes"**;  **function** *Assignment6*() {  **return** (  <**div**>  <**h1**>Assignment 6</**h1**>  <**Classes**/>  </**div**>  );  }  **export default** *Assignment6*; |  |

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.

| ***a6/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.

|  |  |
| --- | --- |
| **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 ***labs/a6/styles*** and then import it into the ***Assignment6*** 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***.

| ***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*; | **import** *Classes* **from "./classes"**;  **import** *Styles* **from "./styles"**;  **function** *Assignment6*() {  **return** (  <**div**>  <**h1**>Assignment 6</**h1**>  <**Styles**/>  <**Classes**/>  </**div**>  )  }  **export default** *Assignment6*; |
|  |

## 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 ***src/labs/a6/conditional-output/conditional-output-if-else.js*** with the following code.

| ***src/labs/a6/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 shortcircuits the content if its false or evaluates the expression if it's true. Implement the equivalent component below in ***src/labs/a6/conditional-output/conditional-output-inline.js***.

| ***src/labs/a6/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 ***labs/index.js***. Confirm all components render as expected.

| ***labs/a6/conditional-output/index.js*** | ***labs/a6/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*; | **import** *Classes* **from "./classes"**;  **import** *Styles* **from "./styles"**;  **import** *ConditionalOutput*  **from "./conditional-output"**;  **function** *Assignment6*() {  **return** (  <**div**>  <**h1**>Assignment 6</**h1**>  <**ConditionalOutput**/>  <**Styles**/>  <**Classes**/>  </**div**>  );  }  **export default** *Assignment6*; |
|
| Assignment 6Welcome If ElsePlease login Inline | |

## 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 ***src/labs/a6/todo***, implement the ***TodoItem*** complement in a ***todo-item.js*** file as shown below. Import the component into the ***Assignment6*** component and confirm that it renders as shown.

| ***src/labs/a6/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*; |  |

Refactor ***todos.js*** into a JSON file as shown below

| ***src/labs/a6/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 re-implement the ***todo-list.js*** as shown below. Import ***todo-list.js*** in ***labs/a6/index.js***, refresh the browser, and confirm the ***TodoList*** renders a list of checkboxes and todo items.

| ***src/labs/a6/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*; |  |

# 3 Tuiter

Now that you've had a chance to practice creating React.js components, let's apply these skills to port the ***Tuiter*** clone implementation from previous assignments. We'll re-implement tuiter as a set of React.js components under ***src/tuiter***. Let's start with re-implementing components that have little or no dependencies on other components.

## 3.1 Re-implementing NavigationSidebar as a React component

The ***NavigationSidebar*** component created in a previous assignment is a good place to start because it has no dependencies on other components, so we should be able to re-implement it easily and test it independently. Under the ***src/tuiter*** directory, create all your new components starting with ***NavigationSidebar***. Implement the component in src/tuiter/navigation-sidebar/index.js as shown below. We will walk you step by step on how to re-implement the first couple of components and as you gain confidence on how to do this, we will progressively give you less detailed instructions and let you do it on your own.

### 3.1.1 Importing React.js

The first thing you need to do is to import the ***React.js*** library. All React components must at least import this one library. Open the ***index.js*** and import ***React.js***.

| ***src/tuiter/navigation-sidebar/index.js*** | |
| --- | --- |
| **import *React* from "react"**;  **const** NavigationSidebar = () => {  **return null**;  };  **export default** *NavigationSidebar*; | |

### 3.1.2 Object deconstructed parameter

Convert all parameters into an object deconstructor and provide initial default values.

| ***src/tuiter/navigation-sidebar/index.js*** | |
| --- | --- |
| **import *React* from "react"**;  **const** *NavigationSidebar* = (  {  active = **'explore'**  }  ) => {  **return null**;  };  **export default** *NavigationSidebar*; | |

### 3.1.3 React function components return a single HTML element

React.js function components can only return a single HTML element. If the function needs to return more than one element, then they need to be wrapped with a parent element. Most common element is a simple DIV as shown below. Alternatively you can use fragment syntax which looks like tags with no names ***<></>***

| ***src/tuiter/navigation-sidebar/index.js*** | |
| --- | --- |
| **return** (  <**div**>  </**div**>  ); | |

### 3.1.4 Use class instead of className

The HTML ***class*** attribute is commonly used to associate CSS transformation rules to an HTML element. In JavaScript ***class*** is a keyword so we can't use it. Instead we use ***className*** which will be transpiled into ***class*** in the resulting DOM. Replace all ***class*** attributes with ***className***.

| ***src/tuiter/navigation-sidebar/index.js*** | |
| --- | --- |
| **return** (  <**div className="list-group"**>  <**a className="list-group-item"**>Tuiter</**a**>  <**a className="list-group-item"**>Home</**a**>  <**a className="list-group-item"**>Explore</**a**>  <**a className="list-group-item"**>Notifications</**a**>  <**a className="list-group-item"**>Messages</**a**>  <**a className="list-group-item"**>Bookmarks</**a**>  <**a className="list-group-item"**>Lists</**a**>  <**a className="list-group-item"**>Profile</**a**>  <**a className="list-group-item"**>More</**a**>  </**div**>  ); | |

### 3.1.5 Render dynamic classes with embedded expressions

If an attribute value is dynamic then use string template expressions to dynamically style the component as shown below. The ***active*** parameter is used to dynamically choose which link to display as active.

| ***src/tuiter/navigation-sidebar/index.js*** |  |
| --- | --- |
| **import *React* from "react"**;  **const** *NavigationSidebar* = (  {  active = **'explore'**  }  ) => {  **return** (  <**div className="list-group"**>  <**a className="list-group-item"**>Tuiter</**a**>  <**a className=**{**`list-group-item**  ${active === **'home'**?**'active'**:**''**}**`**}>  Home  </**a**>  <**a className=**{**`list-group-item**  ${active === **'explore'**?**'active'**:**''**}**`**}>  Explore  </**a**>  <**a className=**{**`list-group-item**  ${active === **'notifications'**?**'active'**:**''**}**`**}>  Notifications  </**a**>  <**a className=**{**`list-group-item**  ${active === **'messages'**?**'active'**:**''**}**`**}>  Messages  </**a**>  <**a className=**{**`list-group-item**  ${active === **'bookmarks'**?**'active'**:**''**}**`**}>  Bookmarks  </**a**>  <**a className=**{**`list-group-item**  ${active === **'lists'**?**'active'**:**''**}**`**}>  Lists  </**a**>  <**a className=**{**`list-group-item**  ${active === **'profile'**?**'active'**:**''**}**`**}>  Profile  </**a**>  <**a className=**{**`list-group-item**  ${active === **'more'**?**'active'**:**''**}**`**}>  More  </**a**>  </**div**>  );  };  **export default** *NavigationSidebar*; |  |

To test the ***NavigationSidebar*** component, import it from ***tuiter.js*** and pass it "***home***" for the ***active*** attribute. Refresh the browser and confirm the sidebar displays as expected. Add the icons as necessary.

| ***src/tuiter/index.js*** |  |
| --- | --- |
| **import** *Nav* **from "../nav"**;  **import** *NavigationSidebar*  **from "./navigation-sidebar"**;  **function** *Tuiter*() {  **return** (  <**div**>  <**Nav**/>  <**NavigationSidebar active="home"**/>  <**h1**>Tuiter</**h1**>  </**div**>  )  }  **export default** *Tuiter* |  |

## 3.2 Re-implement WhoToFollowListItem as a React component

The ***WhoToFollowListItem*** component created in a previous assignment is a good place to continue because it has no dependencies on other components, so we should be able to re-implement it easily and test it independently. In the ***src/tuiter*** directory create folder ***who-to-follow-list*** and re-implement the ***WhoToFollowListItem*** component in ***who-to-follow-list-item.js***. Feel free to re-use the implementation from the previous assignment. Below is an example implementation of the ***WhoToFollowListItem*** component that might be missing some styling. Feel free to re-use it.

| ***src/tuiter/who-to-follow-list/who-to-follow-list-item.js*** | |
| --- | --- |
| **import *React* from "react"**;  **const** *WhoToFollowListItem* = (  {  who = { **userName**: **'NASA'**, **handle**: **'NASA'**, **avatarIcon**: **'nasa.png'** }  }  ) => {  **return**(  <**li className="list-group-item"**>  <**div className="row"**>  <**div className="col-2"**>  <**img className="rounded-circle" height=**{48} **src=**{**`/images/**${who.**avatarIcon**}**`**}/>  </**div**>  <**div className="col-8"**>  <**div className="fw-bold"**>{who.**userName**}</**div**>  <**div**>@{who.**handle**}</**div**>  </**div**>  <**div className="col-2"**>  <**button className="btn btn-primary rounded-pill float-end"**>Follow</**button**>  </**div**>  </**div**>  </**li**>  );  };  **export default** *WhoToFollowListItem*; | |

To test the ***WhoToFollowListItem*** component, import it from ***src/tuiter/index.js*** as shown below. Refresh the browser and confirm the ***NavigationSidebar*** and ***WhoToFollowListItem*** display as expected. Remove it when done testing.

| ***src/tuiter/index.js*** |  |
| --- | --- |
| **import** *Nav* **from "../nav"**;  **import** *NavigationSidebar*  **from "./navigation-sidebar"**;  **import** *WhoToFollowListItem*  **from "./who-to-follow-list/who-to-follow-list-item"**;  **function** *Tuiter*() {  **return** (  <**div**>  <**Nav**/>  <**NavigationSidebar active="home"**/>  <**WhoToFollowListItem**/>  <**h1**>Tuiter</**h1**>  </**div**>  )  }  **export default** *Tuiter* |  |

Try passing a different value for the ***who*** attribute as shown below. Refresh the browser and confirm the ***WhoToFollowListItem*** displays as expected. Remove them when dones testing.

| ***src/tuiter/index.js*** |  |
| --- | --- |
| <**WhoToFollowListItem who=**{{  **userName**: **'NASA'**, **handle**: **'NASA'**,  **avatarIcon**: **'nasa.png'**,  }}/>  <**WhoToFollowListItem who=**{{  **userName**: **'Tesla'**, **handle**: **'tesla'**,  **avatarIcon**: **'tesla.png'**,  }}/>  <**WhoToFollowListItem who=**{{  **userName**: **'SpaceX'**, **handle**: **'SpaceX'**,  **avatarIcon**: **'spacex.png'**,  }}/> |  |

## 3.3 Re-implemting WhoToFollowList as a React component

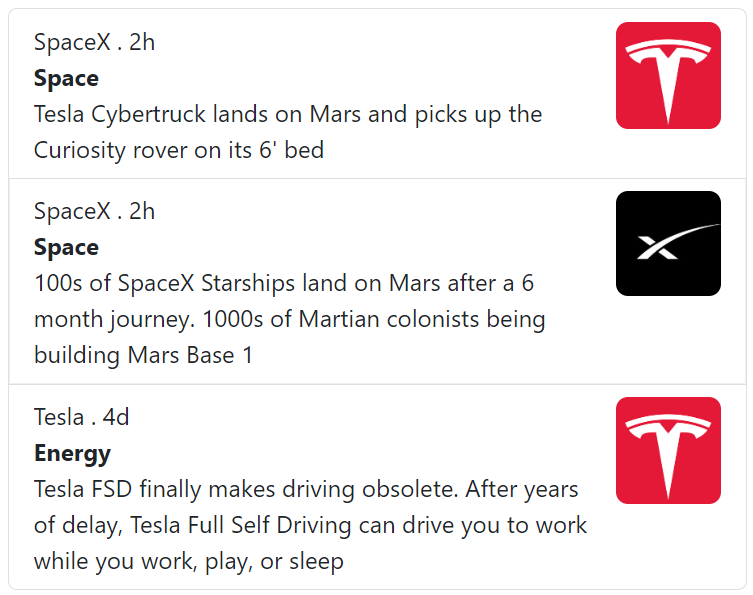
The ***WhoToFollowList*** component built in a previous assignment depended on the ***WhoToFollowListItem*** component loading an array of ***who*** objects and rendering a ***WhoToFollowListItem*** instance for each object in the array. In ***src/tuiter/who-to-follow-list/index.js***, re- implement the ***WhoToFollowList*** component from the previous assignment as a React component. You can use the code below as a guide.

| ***src/tuiter/who-to-follow-list/index.js*** | | ***who.json*** |
| --- | --- | --- |
| **import *React* from "react"**;  **import** whoArray **from './who.json'**;  **import** *WhoToFollowListItem*  **from "./who-to-follow-list-item"**;  **const** *WhoToFollowList* = () => {  **return**(  <**ul className="list-group"**>  <**li className="list-group-item"**>  <**h3**>Who to follow</**h3**>  </**li**>  {  whoArray.map(who =>  <**WhoToFollowListItem**  **key=**{who.**\_id**}  **who=**{who}/>  )  }  </**ul**>  );  };  **export default** *WhoToFollowList*; | | **[**  **{**  **"\_id": 123,**  **"userName": "NASA",**  **"handle": "NASA",**  **"avatarIcon": "nasa.png"**  **},**  **{**  **"\_id": 234,**  **"userName": "Tesla",**  **"handle": "tesla",**  **"avatarIcon": "tesla.png"**  **},**  **{**  **"\_id": 345,**  **"userName": "SpaceX",**  **"handle": "SpaceX",**  **"avatarIcon": "spacex.png"**  **}**  **]** |

## 3.4 Re-implemeting PostSummaryItem as a React component

The ***PostSummaryItem*** implemented in the previous assignment is very similar to the ***WhoToFollowListItem***. In ***src/tuiter/post-summary-list/post-summary-item.js***, re-implement the ***PostSummaryItem*** from the previous assignment as a React component. You can use the following code as a guide.

| ***src/tuiter/post-summary-list/post-summary-item.js*** | |
| --- | --- |
| **import *React* from "react"**;  **const** *PostSummaryItem* = (  {  post = {  **"topic"**: **"Space"**,  **"userName"**: **"SpaceX"**,  **"time"**: **"2h"**,  **"title"**: **"Tesla Cybertruck lands on Mars and picks up the Curiosity rover on its 6' bed"**,  **"image"**: **"tesla.png"**  }  }  ) => {  **return**(  <**li className="list-group-item"**>  <**div className="row"**>  <**div className="col-10"**>  <**div**>{post.**userName**} . {post.**time**}</**div**>  <**div className="fw-bolder"**>{post.**topic**}</**div**>  <**div**>{post.**title**}</**div**>  </**div**>  <**div className="col-2"**>  <**img width=**{70} **className="float-end rounded-3" src=**{**`/images/**${post.**image**}**`**}/>  </**div**>  </**div**>  </**li**>  );  };  **export default** *PostSummaryItem*; | |

To test the ***PostSummaryItem*** component you can import it from ***src/tuiter/index.js*** like we did to test the ***WhoToFollowListItem*** and pass in an object for the ***post*** attribute. Refresh the browser and confirm the ***PostSummaryItem*** displays as expected. Try passing a different value for the ***post*** attribute. Refresh the browser and confirm the ***PostSummaryItem*** displays as expected.

## 3.5 Re-implementing PostSummaryList with React

The ***PostSummaryList*** implemented in the previous assignment is very similar to the ***WhoToFollowList***. In ***src/tuiter/post-summary-list/ index.js***, re-implement the ***PostSummaryList*** from the previous assignment as a React component. You can use the following code as a guide. Import the component in your ***Tuiter*** component and confirm it renders as shown here on the right.

| ***src/tuiter/post-summary-list/index.js*** | ***posts.json*** |
| --- | --- |
| **import *React* from "react"**;  **import** postsArray **from './posts.json'**;  **import** *PostSummaryItem*  **from "./post-summary-item"**;  **const** *PostSummaryList* = () => {  **return**(  <**ul className="list-group"**>  {  postsArray.map(post =>  <**PostSummaryItem**  **key=**{who.**\_id**} **post=**{post}/> )  }  </**ul**>  );  };  **export default** *PostSummaryList*; | [{ **"\_id"**: **123**, **"topic"**: **"Space"**, **"userName"**: **"SpaceX"**,  **"title"**: **"Tesla Cybertruck lands on Mars and picks up the Curiosity rover on its 6' bed"**,  **"time"**: **"2h"**, **"image"**: **"tesla.png"** },  { **"\_id"**: **234**, **"topic"**: **"Space"**, **"userName"**: **"SpaceX"**,  **"title"**: **"100s of SpaceX Starships land on Mars after a 6 month journey. 1000s of Martian colonists being building Mars Base 1"**,  **"time"**: **"2h"**, **"image"**: **"spacex.png"** },  { **"\_id"**: **345**, **"topic"**: **"Energy"**, **"userName"**: **"Tesla"**,  **"title"**: **"Tesla FSD finally makes driving obsolete. After years of delay, Tesla Full Self Driving can drive you to work while you work, play, or sleep"**,  **"time"**: **"4d"**, **"image"**: **"tesla.png"** }  ] |

## 3.6 Re-implementing ExploreComponent as a React component

The ***ExploreComponent*** implemented in the previous assignment consisted of the main content of the explore screen. In ***src/tuiter/explore/index.js***, re-implement the ***ExploreComponent*** from the previous assignment as a React component. You can use the following code as a guide. Import the component in your ***Tuiter*** component and confirm it renders as shown here on the right.

| ***src/tuiter/explore/index.js*** | | ***index.css*** |
| --- | --- | --- |
| **import *React* from "react"**;  **import** *PostSummaryList* **from "../post-summary-list"**;  **import "./index.css"**;  **const** *ExploreComponent* = () => {  **return**(  <>  <**div className="row"**>  <**div className="col-11 position-relative"**>  <**input placeholder="Search Tuiter"**  **className="form-control rounded-pill ps-5"**/>  <**i className="bi bi-search position-absolute**  **wd-nudge-up"**></**i**>  </**div**>  <**div className="col-1"**>  <**i className="wd-bottom-4 text-primary float-end bi**  **bi-gear-fill fs-2 position-relative"**></**i**>  </**div**>  </**div**>  <**ul className="nav nav-pills mb-2"**>  <**li className="nav-item"**>  <**a className="nav-link active"**>For You</**a**>  </**li**>  <**li className="nav-item"**>  <**a className="nav-link"**>Trending</**a**>  </**li**>  <**li className="nav-item"**>  <**a className="nav-link"**>News</**a**>  </**li**>  </**ul**>  <**div className="position-relative mb-2"**>  <**img src="/images/starship.jpg" className="w-100"**/>  <**h1 className="position-absolute wd-nudge-up text-white"**>  SpaceX Starship</**h1**>  </**div**>  <**PostSummaryList**/>  </>  );  };  **export default** *ExploreComponent*; | | **.wd-tabs li {**  **display: inline;**  **}**  **.wd-tabs {**  **padding-left: 0px;**  **}**  **.wd-selected {**  **border-bottom: 2px blue solid;**  **padding-bottom: 20px;**  **}**  **.wd-nudge-up {**  **bottom: 18px;**  **left: 25px;**  **}**  **.wd-bottom-2 {**  **bottom: 2px;**  **}**  **.wd-bottom-4 {**  **bottom: 4px;**  **}** |

## 3.7 Implement the Tuiter React component

Finalling the ***Tuiter*** component brings all the other components into a single screen. In ***src/tuiter/index.js***, implement the ***Tuiter*** to render ***NavigationSidebar*** on the left hand side, the ***WhoToFollowList*** component on the right hand side and the ***ExploreComponent*** in the center of the screen. Below is an example of how the component might be implemented. Feel free to use the code as a guide and modify as necessary.

| ***src/tuiter/index.js*** | |
| --- | --- |
| **import *React* from "react"**;  **import** *ExploreComponent* **from "./explore"**;  **import** *NavigationSidebar* **from "./navigation-sidebar"**;  **import** *WhoToFollowList* **from "./who-to-follow-list"**;  **function** *Tuiter*() {  **return** (  <**div className="row mt-2"**>  <**div className="col-2 col-md-2 col-lg-1 col-xl-2"**>  <**NavigationSidebar active="explore"**/>  </**div**>  <**div className="col-10 col-md-10 col-lg-7 col-xl-6"**  **style=**{{**"position"**: **"relative"**}}>  <**ExploreComponent**/>  </**div**>  <**div className="d-sm-none d-md-none d-lg-block col-lg-4 col-xl-4"**>  <**WhoToFollowList**/>  </**div**>  </**div**>  );  }  **export default** *Tuiter* | |
|  | |

# 

# 4 Challenge (required for graduate students)

## 4.1 Re-implement the HomeScreen as React component

In a new folder called ***scr/tuiter/home***, re-implement the JavaScript ***HomeComponent*** component implemented in an earlier assignment as a React.js component. Use the wireframe here on the right as a guide. Fell free to use your own content, images, or re-use the HTML from previous assignments.

## 4.2 Refactor navigation

The ***Tuiter*** (***src/tuiter/index.js***) component currently just renders the ***ExploreComponent*** in the main content in the center column. Refactor the Tuiter component to display the ***HomeComponent*** or the ***ExploreComponent*** when users navigate to either ***/tuiter/home*** or ***/tuiter/explore*** respectively. This can be accomplished by declaring nested routes in the Tuiter component to display the Home component or Explore component based on the route as shown below. Use the code below for guidance.

| ***src/tuiter/index.js*** | |
| --- | --- |
| **import *ExploreComponent* from "./explore";**  **import *NavigationSidebar* from "./navigation-sidebar";**  **import *WhoToFollowList* from "./who-to-follow-list";**  **import {*Routes*, *Route*} from "react-router";**  **import *HomeComponent* from "./home";**  **function** *Tuiter*() {  **return** (  <**div className="row mt-2"**>  <**div className="col-2 col-md-2 col-lg-1 col-xl-2"**>  <**NavigationSidebar active="explore"**/>  </**div**>  <**div className="col-10 col-md-10 col-lg-7 col-xl-6"**  **style=**{{**"position"**: **"relative"**}}>  <**Routes**>  <**Route path="home" element=**{<**HomeComponent**/>}/>  <**Route path="explore" element=**{<**ExploreComponent**/>}/>  </**Routes**>  </**div**> | |

For the nested routes to work, you'll have to redefine the Tuiter route in App.js as shown below

| ***src/App.js*** | |
| --- | --- |
| <**Routes**>  <**Route index element=**{<**Labs**/>}/>  <**Route path="/hello" element=**{<**HelloWorld**/>}/>  <**Route path="/tuiter/\*" element=**{<**Tuiter**/>}/>  </**Routes**> | |

Restart the application and confirm that navigating to http://localhost:3000/tuiter/home displays the ***HomeComponent*** and navigating to http://localhost:3000/tuiter/ explore displays the ***ExploreComponent***.

Now, refactor ***NavigationSideBar*** hyperlinks to use ***Link*** elements instead. Use the ***to*** attributes to point to the ***HomeScreen*** and ***ExploreScreen*** paths as shown below. Add a ***Labs*** link to to navigate back to navigate back to the labs. Instead of highlighting the links using the ***active*** parameter, use the ***useLocation()*** hook to parse the URL and extract the location to highlight the corresponding link. Use the code below for guidance.

| ***src/tuiter/navigation-sidebar/index.js*** | |
| --- | --- |
| **import *React* from "react"**;  **import** {*Link*} **from "react-router-dom"**;  **import** {*useLocation*} **from "react-router"**;  **const** *NavigationSidebar* = () => {  **const** {pathname} = *useLocation*();  **const** paths = pathname.split(**'/'**)  **const** active = paths[2];  **return** (  <**div className="list-group"**>  <**a className="list-group-item"**>Tuiter</**a**>  <**Link to="/tuiter/home" className=**{**`list-group-item** ${active === **'home'**?**'active'**:**''**}**`**}>  Home  </**Link**>  <**Link to="/tuiter/explore" className=**{**`list-group-item** ${active === **'explore'**?**'active'**:**''**}**`**}>  Explore  </**Link**>  <**Link to="/" className="list-group-item"**>  Labs  </**Link**>  ...  <**a className=**{**`list-group-item** ${active === **'more'**?**'active'**:**''**}**`**}>  More  </**a**>  </**div**>  );  };  **export default** *NavigationSidebar*; | |

# 5 Deliverables

As a deliverable, make sure you complete the ***Labs***, ***Tuiter*** and ***Challange*** (if graduate student) sections of this assignment. All your work must be done in a branch called ***a6***. When done, add, commit and push the branch to GitHub. Deploy the new branch to Netlify and confirm it's available in a new URL based on the branch name. Submit the link to your GitHub repository and the new URL where the branch deployed to in Netlify. Here's an example on the steps:

| *Create a branch called* ***a6*** |
| --- |
| git checkout -b a6  # do all your work |

Do all your work, e.g., ***Labs*** exercises, ***Tuiter***, ***Challenge*** (graduate students)

| *Add, commit and push the new branch* |
| --- |
| git add .  git commit -am "a6 React.js sp22"  git push |

If you have ***Netlify*** configured to auto deploy, then confirm it auto deployed. If not, then deploy the branch manually.

In Canvas, submit the following

1. The new URL where your ***a6*** branch deployed to on Netlify
2. The link to your new branch in GitHub.