assignment-4-bank-of-react-spring23.md

Assignment 4 - Bank of React

DUE DATE

- **Due on Friday, 4/14, at 11:59 PM**
- **You can work in groups or individually. Maximum 4 students per group. Working in groups is strongly recommended and encouraged.**

HOW TO SUBMIT

- **Blackboard:** Submit (1) the link to your GitHub repository, (2) the link to your GitHub Pages website, and (3) your group member names on Blackboard by 11:59 PM on the due day. Each student must submit the assignment individually on Blackboard—even when working in a group.
- **GitHub:** In the README section of GitHub repository, please list your group member names and GitHub usernames.
- **GitHub Pages:** Deploy your website to GitHub Pages (instructions on deploying a React application to GitHub Pages and additional instructions). In the README section of GitHub repository, please include a link to the GitHub Pages

website.

GRADING

This assignment is worth **20%** of your grade.

- **15%** - Assignment functionality
- **4%** - Code documentation (i.e., proper commenting). Code organization (i.e., the code is clean, well-formatted, and easy-to-read). Git version control such as following Git feature branch workflow, creating pull requests when merging feature branches, making small and frequent commits with appropriate commit messages, etc.
- **1%** - Deploy the website to GitHub Pages (instructions on deploying a React application to GitHub Pages and additional instructions)

GOAL

The goal is to offer students the opportunity to gain experience in React application development by:

- Learning about client-side routing
- Adding client-side routing to React applications using React Router
- Setting up React Router to enable common browser behavior
- Utilizing built-in React Router components (Route, Router, Link, and Redirect)
- Working with common React features such as components, state, props, event handlers, etc.

TECHNICAL BACKGROUND

Before we discuss the Assignment in a later section, this technical section introduces client-side routing, Route, Router, Link, and Redirect. It also creates an example website to serve as the starter code of the Assignment. Example (starter) code for Assignment 4

Client-Side Routing

On the server, `routing` (i.e., server-side `routing`) generally refers to the way we define the URLs and RESTful resources that make up our application. Whether we are retrieving data from the database or storing data, our server needs to know where the data lives. Server-side `routes` can help us keep track of this information.

On the client, i.e., in the web browser, things are a little different. When we build a Single-Page Application (SPA) using React, we create only a single HTML page (index.html) to render data inside of the web browser. Client-side `routing` is the technique used to give users the impression of navigating between multiple web pages, by fetching different data from the server and rendering different `views` in the same single HTML page. This process is done on the client using JavaScript. Client-side `routing` is how we describe which `views` are displayed on the web page at any given time.

React Router

Although the basic React library doesn't have the capability to handle client-side `routing`, there are other libraries available to handle this specific task. The most popular library for client-side `routing` is called React Router. Checkout the official documentation.

React Router allows us to use familiar `routing` patterns instead of writing complicated `if-statements` in JavaScript.

Implementing Client-Side Routing

"Bank of React" Application

To demonstrate the power of React Router and client-side `routing`, we are going to build a simple personal banking application called **Bank of React**, where we can independently display the Credits and Debits made to each bank account.

The bank website consists of the following web page `views` that render different data:

- Home
- User Profile

- Login
- Credits
- Debits

We are going to implement client-side `routing` to display different web page `views`. When users visit the bank website, they will be greeted by the Home page `view`. From there, they can navigate to different web page `views`; e.g., User Profile, Login, Credits, Debits, and of course Home.

Purpose

In the following sections, we will build out parts of the **Bank of React** application, such as Home, User Profile, and Login page `views`. During the process, we will discuss some key features of React and create components that can be applied in Assignment 4.

In Assignment 4, you will implement the Credits and Debits page `views`.

Creating a React Application

Before we jump into React Router, let's create a new React application for the **Bank of React**:

```
npx create-react-app bank-of-react
```

Let's start up our new React application and make sure everything looks good before we move on. Go to the project directory and run the application using the following commands:

```
cd bank-of-react
npm start
```

Inside the `App` component, we clear out the React boilerplate and replace it with "Hello, World!":

```
// src/App.js

import React, {Component} from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        Hello, World!
      </div>
    );
  }
}

export default App;
```

If everything looks good, we can proceed with creating the **Bank of React** application.

Stop the React application. First, we need to stop the React application so that we can modify it to create the **Bank of React** application. Note: To stop the application, press `CTRL` and `C` keys in the terminal.

Installing React Router

React Router is packaged in `npm`. We can use `npm` to pull the React Router library code into our application. The package is `react-router-dom`.

We will install the React Router library and other dependencies:

```
npm install
```

Note: The Bank of React application uses React Router version 5 and should be compatible with later versions of React Router.

If you encounter problems with a later version of React Router, you can specifically install React Router version 5 using the following command (you can install this version locally in the project directory):

```
npm install react-router-dom@5.3.0 react-router@5.2.1
```

Setting Up the Router

- Now that we have installed the React Router library, we will create a `Router` in our application. Because React is not a browser-only framework, the React Router team provides sepcific versions of `Router` for different environments. For our browser-rendered React application, we will use the `BrowserRouter` of `react-router-dom` library.
- In the `App.js` file, we import the `BrowserRouter` library and make an instance (alias) of it as a component called `Router`.
- Once we have imported the `BrowserRouter` library (as `Router`), we use the `Router` component (`<Router />`) to wrap the top-level React component, as follows:

```
// src/App.js

import React, {Component} from 'react';
import {BrowserRouter as Router} from 'react-router-dom';

class App extends Component {
  render() {
    return (
      <Router>
        <div className="App">
          Hello, World!
        </div>
      </Router>
    );
  }
}
```

```
export default App;
```

- The `Router` component is used to wrap all other `route` components for different page `views`.
- Note: `Router` component (i.e., `BrowserRouter`) is a parent component and can only have one single `<div>` child component.

Creating a React Component (Home)

- Let's create a `Home` component to be the Home page `view`. We create a `Home.js` file to contain the `Home` component:

```
// src/components/Home.js

import React, {Component} from 'react';

class Home extends Component {
  render() {
    return (
        <div>
          <img src="https://picsum.photos/200/200" alt="bank"/>
          <h1>Bank of React</h1>
        </div>
    );
  }
}

export default Home;
```

Setting Up a Route within the Router

- Now that we have the `Home` component, we want to display it on the application's Home page instead of the text string "Hello, World!".

- To do that, we assign a `Route` to the `Home` component for the Home page `view`.
- We import another library called `Route` from `react-router-dom` and mount it within the `App.js`. That is, `App.js` contains all routes of our React application.
- Here is the `App.js` file:

```
// src/App.js

import React, {Component} from 'react';
import {BrowserRouter as Router, Route} from 'react-router-dom';
import Home from './components/Home';

class App extends Component {
  render() {
    return (
    <Router>
      <div>
        <Route exact path="/" component={Home}/>
      </div>
    </Router>
    );
  }
}

export default App;
```

- If we check the web browser at `http://localhost:3000`, we should now see the Home page `view`.
- Let's break down the syntax:

```
// src/components/App.js

import {BrowserRouter as Router, Route} from 'react-router-dom';
```

```
...
<Router>
  <div>
    <Route exact path="/" component={Home}/>
  </div>
</Router>
...
```

- We import the libraries of `BrowserRouter` (as `Router`) and `Route` from `react-router-dom`.
- Note: Adding a `Route` to our application is like mounting any other components. All we have to do is to import it and mount it with some props passed to it.
- We use two props for the `Route` above:
  - The `path` prop specifies which `route` (i.e., URL path) the `Home` component is located.
  - The `component` prop tells the `Route` to mount a `Home` component when the web browser receives a request to *exactly* the '/' `route`.
- The `exact` keyword is very important. With it, the web browser won't show the `Home` page `view` if there are any other paths appended after the initial '/' `route`. For example, the '/credits' `route` won't show the `Home` page `view` because it is not *exactly* the '/' `route`.
- The Home page `view` can only be displayed when the URL path is *exactly* `http://localhost:3000/`.

Using Props and Route

- On the Home page `view`, we want to see the account balance. We will add some extra parts to the `Home` component for displaying the account balance.
- Let's create a re-usable `AccountBalance` component that receives the account balance `accountBalance` as a prop and displays it.
- We create the `AccountBalance` component as follows:

```
// src/components/AccountBalance.js

import React, {Component} from 'react';

class AccountBalance extends Component {
  render() {
    return (
        <div>
          Balance: {this.props.accountBalance}
        </div>
    );
  }
}

export default AccountBalance;
```

- We will include the `AccountBalance` component in the `Home` component, so that we can see the account balance on the Home page `view`.
- In the `Home` component, we render the `AccountBalance` component and pass the `accountBalance` value as a prop to it.

```
// src/components/Home.js

import React, {Component} from 'react';
import AccountBalance from './AccountBalance';

class Home extends Component {
  render() {
    return (
        <div>
          <img src="https://picsum.photos/200/200" alt="bank"/>
          <h1>Bank of React</h1>

          <AccountBalance accountBalance={this.props.accountBalance}/>
```

```
            </div>
        );
      }
    }

    export default Home;
```

Passing State Property as Prop

- Since we want to use the same `accountBalance` value in other parts of the application, we will keep it at the top-level `App` component (in `App.js`) using the `state` property called `accountBalance`, so that it can be passed down to other components in the application.
- In the `App` component, we create the `state` property of `accountBalance` and initialize it with an initial value of 1234567.89 as follows:

```
// src/App.js

class App extends Component {
...
  constructor() {
      super();
      this.state = {
        accountBalance: 1234567.89
      };
    }
...
}
```

- By defining the `state` property `accountBalance` in the top-level `App` component, the `accountBalance` value can now be passed down to other components as a prop using `this.state.accountBalance`.
- A component that receives the `accountBalance` prop can access its value using `this.props.accountBalance`.

- In `App.js`, we specify the '/' `Route` for the Home page `view`, and pass the `accountBalance` value as a prop when rendering the `Home` component as follows:
  - First, assign the `Home` component to the variable `HomeComponent` and pass the `accountBalance` value as a prop to it.
  - Then, pass the `HomeComponent` variable to the '/' `Route` as the `render` prop.

```
// src/App.js

...
render() {
  const HomeComponent = () => (<Home accountBalance={this.state.accountBalance}/>);

  return (
    <Router>
      <div>
        <Route exact path="/" render={HomeComponent}/>
      </div>
    </Router>
  );
}
...
```

- Now we can refresh the web page and see the Home page `view` with the account balance.
- Note: You may notice the different uses of `render` prop and `component` prop in `<Route />`:

  `<Route exact path="/" render={HomeComponent}/>` vs. `<Route exact path="/" component={Home}/>`.

  - If you need to pass `props` to a `Route`, you must use the `render` prop. As shown above, we pass the `accountBalance` prop to `HomeComponent` before we use it in the Home page '/' `Route`.
  - If you don't need to pass `props` to a `Route`, you can use the `component` prop since it cannot pass `props`. In the previous section, we use the `Home` component in the Home page '/' `Route`, where `Home` doesn't take in any

`props` .

Creating Another Component and Route (User Profile)

- Next, let's build out the User Profile page `view` and create a `Route` for it in `App.js` .
- We create the `UserProfile` component as follows:

```
// src/components/UserProfile.js
// The UserProfile component is used to demonstrate the use of Route.

import React, {Component} from 'react';

class UserProfile extends Component {
  render() {
    return (
        <div>
          <h1>User Profile</h1>

          <div>Username: {this.props.userName}</div>
          <div>Member Since: {this.props.memberSince}</div>
        </div>
    );
  }
}

export default UserProfile;
```

- Similar to the account balance, we will keep the values of "Username" and "Member Since" at the top-level `App` component (in `App.js`), using the `state` properties called `userName` and `memberSince` , respectively. This way, they can be passed down to other components in the application.
- In the `App.js` code shown below, we create the `state` properties called `userName` and `memberSince` and initialize them.

- We pass the `userName` and `memberSince` values to the `UserProfile` component as props using `this.state.currentUser.userName` and `this.state.currentUser.memberSince`, respectively.
- As a result, the `UserProfile` component can access the props by using `this.props.userName` and `this.props.memberSince`.
- In addition, we create the '/userProfile' `Route` for the `UserProfile` component (i.e., the User Profile page `view`).

```
// src/App.js

import React, {Component} from 'react';
import {BrowserRouter as Router, Route} from 'react-router-dom';
import Home from './components/Home';
import UserProfile from './components/UserProfile';

class App extends Component {
  constructor() {  // Create and initialize state
    super();
    this.state = {
      accountBalance: 1234567.89,
      currentUser: {
        userName: 'Joe Smith',
        memberSince: '11/22/99',
      }
    };
  }

  // Create Routes and React elements to be rendered using React components
  render() {
    const HomeComponent = () => (<Home accountBalance={this.state.accountBalance}/>);
    const UserProfileComponent = () => (
      <UserProfile userName={this.state.currentUser.userName} memberSince={this.state.currentUser.memb
    );

    return (
        <Router>
```

```
            <div>
              <Route exact path="/" render={HomeComponent}/>
              <Route exact path="/userProfile" render={UserProfileComponent}/>
            </div>
          </Router>
      );
    }

}

export default App;
```

- If we type `http://localhost:3000/userProfile` into the web browser, we see the new User Profile page `view` on the screen.

Setting Up Links Between Components (from Home to User Profile)

- Now that we have set up the `Home` and `UserProfile` components and their routes, we want to have the ability of navigating between these two page `views`.
- For easy navigation, we will add a link from the `Home` component to the `UserProfile` component, so that users can navigate to the User Profile page `view` from the Home page `view`.
- React Router provides a very simple solution for this: using the `Link` library from `react-router-dom`.
- We import the `Link` library and include a `Link` component pointing to the '/userProfile' `Route` as follows:

```
// src/components/Home.js
// The Home component is used to demonstrate the use of Link.

import React, {Component} from 'react';
import AccountBalance from './AccountBalance';
import {Link} from 'react-router-dom';
```

```
class Home extends Component {
  render() {
    return (
        <div>
          <img src="https://picsum.photos/200/200" alt="bank"/>
          <h1>Bank of React</h1>

          <Link to="/userProfile">User Profile</Link>

          <AccountBalance accountBalance={this.props.accountBalance}/>
        </div>
    );
  }
}

export default Home;
```

- React Router's `Link` component takes a prop `to=` that indicates where to redirect.
- Note: We can link to any component, even within child components, as long as the component's `Route` has been initialized in the tree of components (in `App.js`).

Setting Up More Links Between Components (from User Profile back to Home)

- We are now able to navigate to the `UserProfile` component through the link in `Home` component, but we are left stranded once we get there.
- Let's improve the user experience by adding a link back to the `Home` component from the `UserProfile` component.
- This way, users can navigate back and forth between the Home and User Profile page `views`.
- Here is the `Link` component that links back to the Home page `view`:

```
// src/components/UserProfile.js
// The UserProfile component is used to demonstrate the use of Route and Link.

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

```
    ...

    render() {
      return (
        ...

        <Link to="/">Return to Home</Link>
      );
    }
```

Programmatic Redirect (Login)

While the `Link` component is powerful, it requires a user interaction to navigate to another `route`. There are plenty of use cases that require a redirect after a user clicks on a button. One common example is to redirect after a form is submitted. To do this, we will use the `Redirect` library from `react-router-dom`.

We add a mock Login page `view` to the bank website, and then create the `LogIn` component to implement the `Redirect` component.

- In the following code, we create the `LogIn` component (`Login.js`) to demonstrate how the `Redirect` component works, using a event handler function, a set state method, and a rendering method.

```
// src/components/Login.js
// The LogIn component is used to demonstrate the use of Redirect.

import React, { Component } from 'react'
import { Redirect } from 'react-router-dom'

class LogIn extends Component {
  constructor () {  // Create and initialize state
    super();
    this.state = {
      user: {
```

```
        userName: '',
        password: ''
      },
      redirect: false  // Redirect property used to trigger Redirect
    };
  }

  // When the user name input changes, capture the input and update the state (user.userName)
  handleChange = (e) => {
    const updatedUser = {...this.state.user};  // Create an object for state's user properties
    updatedUser.userName = e.target.value;  // Set object's userName to the new input value
    this.setState({user: updatedUser})  // Update state with object values
  }

  // When user clicks on "Log In" button, store user data and then redirect to "User Profile" page
  handleSubmit = (e) => {
    e.preventDefault()
    this.props.mockLogIn(this.state.user)
    this.setState({redirect: true})  // Update state to trigger Redirect
  }

  render () {
    if (this.state.redirect) {  // Redirect to "User Profile" page
      return (<Redirect to="/userProfile"/>);
    }
    // Render the login form
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <div>
            <label>User Name</label>
            <input type="text" name="userName" onChange={this.handleChange} />
          </div>
          <div>
            <label>Password</label>
            <input type="password" name="password" />
          </div>
```

```
            <button>Log In</button>
          </form>
        </div>
      );
    }
  }

  export default LogIn;
```

- Next, in `App.js`, we create the '/login' `Route` for `LogIn` component (i.e., the Login page `view`) as shown below.

Passing a Function as Prop

- In addition to passing a value to a component as a prop, React supports passing a function as a prop.
- In the following `Appl.js` code, we pass the "mockLogIn" function as a prop to the `LogIn` component, so that the `LogIn` component uses the "mockLogIn" function to update the `state` "current user" property in `App.js`.

```
// src/App.js

import LogIn from './components/Login';

...
class App extends Component {
  constructor() {  // Create and initialize state
    super();
    this.state = {
      accountBalance: 1234567.89,
      currentUser: {
        userName: 'Joe Smith',
        memberSince: '11/22/99',
      }
    };
  }

  ...
```

```
  mockLogIn = (logInInfo) => {  // Update state's currentUser (userName) after "Log In" button is clicked
    const newUser = {...this.state.currentUser};
    newUser.userName = logInInfo.userName;
    this.setState({currentUser: newUser})
  }

  ...
  render() {  // Create Routes and React elements to be rendered using React components
    ...
    const LogInComponent = () => (<LogIn user={this.state.currentUser} mockLogIn={this.mockLogIn} />)  //
    ...
    return (
        <Router>
          <div>
            ...
            <Route exact path="/login" render={LogInComponent}/>
            ...
          </div>
        </Router>
    );
  }
}
...
```

- To finish the work on `LogIn` component, we add its link in `Home.js` as shown below, so that users can navigate from the Home page `view` to the Login page `view`.

```
// src/components/Home.js

...
class Home extends Component {
  render() {
    return (
        <div>
          ...
```

```
            <br/>
            <Link to="/login">Login</Link>
            ...
          </div>
      );
    }
  }

  export default Home;
```

The above sections provide a basic set of code for Assignment 4 - Bank of React.

ASSIGNMENT

In this Assignment, you are going to utilize the starter code and enhance it by adding two new pages for Credits and Debits, respectively, as described in the following sections. This is the link to the example (starter) code for Assignment 4.

Tasks: Create the Credits and Debits Pages

For Assignment 4, you will create two new components for displaying the Credits page `view` and the Debits page `view`.

Implementation Suggestions

Your `App` component (in `App.js`) should have:

- `state` properties that include `accountBalance` (number), `credits` (array), and `debits` (array)
- two functions called `addCredit` and `addDebit` that update the `state` based on user input of new credits and debits
- lifecycle method `componentDidMount()` which should include the API requests using the following endpoints:

- Credits API endpoint located at `https://johnnylaicode.github.io/api/credits.json`
- Debits API endpoint located at `https://johnnylaicode.github.io/api/debits.json`

Reference Codes:

- To help you complete Assignment 4, there are reference codes you can re-use in your implementation. In addition to the above sections, you can find useful reference codes in various React examples (link) covered in this course.

Example (Starter) Code Repository

Example (starter) code for Assignment 4 - Bank of React

Deploying a React Application (Website) to GibHub Pages

The above starter code website (React application) is already deployed to GitHub Pages. For instructions on how to deploy a React application to GitHub Pages, please read the repository's README section.

Requirements and Functionalities

1. Calculating and Updating the Account Balance

```
Making the Account Balance Dynamic:

GIVEN I am on any page that displays the Account Balance
WHEN I view the Account Balance display area
THEN I shall see an Account Balance that accurately represents my total Debits subtracted from my total
Credits
AND I shall be able to see a negative account balance if I have more Debits than Credits
AND all amounts are rounded to 2 decimal places (i.e., 1234567.89)
```

How to Calculate Account Balance:

- Account Balance is the difference between total Credits amount and total Debits amount.
- The mathematical formula is: **`Account Balance = total Credits - total Debits`**

2. Adding Credits

```
Viewing the Credits Page:

GIVEN I am on the Home Page
WHEN I click on 'Credits' link/button
THEN I shall be redirected to the Credits Page
AND I shall see a title of 'Credits' on the web page



Displaying List of Credits:

GIVEN I am on the Credits Page
WHEN I view the Credits display area
THEN I shall see all my Credits displayed in a list, including the Credits retrieved from API endpoint
AND each Credit shall display its description, amount, and date (yyyy-mm-dd)
AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)



Adding Credits:

GIVEN I am on the Credits Page
WHEN I enter a new Credit's description and amount
AND WHEN I click on 'Add Credit' button
THEN I shall see my new Credit description and amount added to the Credits display area with the current
date (yyyy-mm-dd)
AND I shall see my Account Balance updated to reflect the addition of new Credit
AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)
```

```
Viewing the Account Balance on the Credits Page:

GIVEN I am on the Credits Page
WHEN I view the Account Balance display area
THEN I shall see my Account Balance displayed
AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)
```

3. Adding Debits

```
Viewing the Debits Page:

GIVEN I am on the Home Page
WHEN I click on 'Debits' link/button
THEN I shall be redirected to the Debits Page
AND I shall see a title of 'Debits' on the web page


Displaying List of Debits:

GIVEN I am on the Debits page
WHEN I view the Debits display area
THEN I shall see all my Debits displayed in a list, including the Debits retrieved from API endpoint
AND each Debit shall display its description, amount, and date (yyyy-mm-dd)
AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)


Adding Debits:

GIVEN I am on the Debits Page
WHEN I enter a new Debit's description and amount
AND WHEN I click on 'Add Debit' button
THEN I shall see my new Debit description and amount added to the Debits display area with the current
date (yyyy-mm-dd)
```

```
    AND I shall see my Account Balance updated to reflect the addition of new Debit
    AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)



    Viewing the Account Balance on the Debits Page:

    GIVEN I am on the Debits Page
    WHEN I view the Account Balance display area
    THEN I shall see my Account Balance displayed
    AND all amounts are rounded to 2 decimal places (e.g., 1234567.89)
```

v1.5