Full Stack Development Lab Instructions
ENSF381: Lab08

React Fetch

Created by: Mahdi Jaberzadeh Ansari (He/Him)
Schulich School of Engineering
University of Calgary
Calgary, Canada
mahdi.ansari1@ucalgary.ca

Week 10, March 11/13, 2024

Contents

1 Introduction

1.1 Objectives

Lab 8 is designed to extend your practical experience in using React.js. In this lab we are going to get comfortable working with forms and public APIs using React. We will create a React application that interacts with the DummyJSON Products API, displays products, allows users to search, view details, and manage products (add, edit, remove) involves several steps.

The objectives of this lab are to equip you with skills in the following areas:

- **Understanding React Components:** Gain a fundamental understanding of React components, the essential building blocks of any React application. This includes learning about JSX, props, and state, as well as how to create both functional and class components.
- **State Management and Lifecycle Methods:** Focus on managing state within React components and leveraging lifecycle methods. You will discover how to manage data within a component, facilitate data flow between components, and utilize lifecycle methods to manage component rendering and behavior effectively.
- **Building a React Application:** Develop your skills further by constructing a small React application. This will involve setting up the development environment, structuring a component hierarchy, and implementing routing and state management. Through this application, you will demonstrate the core features of React and their integration in real-world applications.

1.2 Prerequisites

1. Basic understanding of computer operations.
2. A web browser (Chrome, Firefox, Safari, etc.) to view your HTML file.
3. GitHub account.

1.3 Forming groups

- In this lab session, you **MUST** work with a partner (groups of three or more are not allowed).
- The main goal of working in a group is to learn:
  - how to do teamwork
  - how to not be a single player
  - how to not be bossing
  - how to play for team
  - how to tolerate others
  - how to behave with colleagues
  - how to form a winner team
  - and ...
- Working with a partner usually gives you the opportunity to discuss some of the details of the topic and learn from each other. Also, it will give you the opportunity to practice one of the popular methods of program development called pair programming. In this method, which is normally associated with the "Agile Software Development" technique, two programmers normally work together on the same workstation (you may consider a Zoom session for this purpose). While one partner, the driver, writes the code, the other partner, acting as an observer, looks over his or her shoulder, making sure the syntax and solution logic are correct. Partners should switch roles frequently in such a way that both have equivalent opportunities to practice both roles. **Please note that you MUST switch roles for each exercise.**

- When you have to work with a partner:
  - Choose a partner that can either increase your knowledge or transfer your knowledge. (i.e., do not find a person with the same programming skill level!)
  - Please submit only one lab report with both names. Submitting two lab reports with the same content will be considered copying and plagiarism.

1.4 Before submission

- For most of the labs, you will receive a DOCX file that you need to fill out the gaps. Make sure you have this file to fill out.
- All your work should be submitted as a single file in PDF format. For instructions about how to provide your lab reports, study the posted document on the D2L called How to Hand in Your Lab assignment.
- Please note that if it is group work, only one team member must submit the solution in D2L. For ease of transferring your marks, please mention the group member's name and UCID in the description window of the submission form.
- If you have been asked to write code (HTML, CSS, JS, etc.), make sure the following information appears at the top of your code:
  - File Name
  - Assignment and exercise number
  - Your names in alphabetic order
  - Submission Date:

  Here is an example for CSS and JS files:

```
<!--
============================================================
Name        : lab8_exe_D.html
Assignment  : Lab 8, Exercise D
Author(s)   : Mahdi Ansari, William Arthur Philip Louis
Submission  : May 21, 2030
Description : React.
============================================================
-->
```

- Some exercises in this lab and future labs will not be marked. Please do not skip them, because these exercises are as important as the others in learning the course material.
- In courses like ENSF381, some students skip directly to the exercises that involve writing code, skipping sections such as "Read This First," or postponing the diagram-drawing until later. That's a bad idea for several reasons:
  - "Read This First" sections normally explain some technical or syntax details that may help you solve the problem or may provide you with some hints.
  - Some lab exercises may ask you to draw a diagram, and most of the students prefer to hand-draw them. In these cases, you need to scan your diagram with a scanner or an appropriate device, such as your mobile phone, and insert the scanned picture of your diagram into your PDF file (the lab report). A possible mobile app to scan your documents is Microsoft Lens, which you can install on your mobile device for free. Please make sure your diagram is clear and readable; otherwise, you may either lose marks or it could be impossible for TAs to mark it at all.
  - Also, it is better to use the Draw.io tool if you need to draw any diagram.

- Drawing diagrams is an important part of learning how to visualize data transfer between modules and so on. If you do diagram-drawing exercises at the last minute, you won't learn the material very well. If you do the diagrams first, you may find it easier to understand the code-writing exercises, so you may be able to finish them more quickly.

- **Due Dates:**
  - You must submit your solution until 11:59 p.m. on the same day that you have the lab session.
  - Submissions until 24 hours after the due date get a maximum of 60% of the mark, and after 24 hours, they will not be evaluated and get 0.

1.5 Academic Misconduct/Plagiarism

- Ask for help, but don't copy.
  - You can get help from lab instructor(s), TAs, or even your classmates as long as you do not copy other people's work.
  - If we realize that even a small portion of your work is a copy from a classmate, both parties (the donor and the receiver of the work) will be subject to academic misconduct (plagiarism). More importantly, if you give exercise solutions to a friend, you are not doing him or her a favor, as he or she will never learn that topic and will pay off for this mistake during the exam or quiz. So, please do not put yourself and your friend in a position of academic misconduct.
  - You can use ChatGPT, but please note that it may provide similar answers for others too, or even the wrong answers. For example, it has been shown that AI can hallucinate, proposing the use of libraries that do not actually exist [1]. So, we recommend that you imagine ChatGPT as an advanced search engine, not a solution provider.
  - In order to find out who is abusing these kinds of tools, we will eventually push you toward the incorrect responses that ChatGPT might produce. In that case, you might have failed for the final mark and be reported to administration.
  - If we ask you to investigate something, don't forget to mention the source of your information. Reporting without reference can lead to a zero mark even by providing a correct answer.

1.6 Marking Scheme

- You should not submit anything for the exercises that are not marked.
- In Table 1, you can find the marking scheme for each exercise.

Table 1: Marking scheme

Exercise	Marks
A	5 marks
B	25 marks
C	20 marks
Total	50 marks

1.7 Complains

- Your grades will be posted one week following the submission date, which means they will be accessible at the subsequent lab meeting.

[1] https://perma.cc/UQS5-3BBP

- Normally, the grades for individual labs are assessed by a distinct TA for each lab and section. Kindly refrain from contacting all TAs. If you have any concerns regarding your grades, please direct an email to the TA responsible for that specific lab.

2 Exercise A (Initiating the React Project)

2.1 Prerequisites

We will assume that you have some familiarity with HTML and JavaScript, but you should be able to follow along even if you are coming from a different programming language. We will also assume that you are familiar with programming concepts like functions, objects, arrays, and to a lesser extent, classes.

If you need to review JavaScript, we recommend reading this guide. Note that we are also using some features from ES6 — a recent version of JavaScript. In this lab, we are using `arrow functions`, `classes`, `let`, and `const` statements.

2.2 Read This First

2.2.1 Axios

Axios is a promise-based HTTP client for node.js and the browser. It can run in the browser and nodejs with the same codebase. On the server-side it uses the native node.js http module, while on the client (browser), it uses XMLHttpRequests. We are going to use Axios to make HTTP requests. For examples:

1. GET Request using AXIOS:

```
axios.get('https://dummyjson.com/users')
  .then(function (response) {
    // handle success
    console.log(response);
  });
```

2. POST Request using AXIOS:

```
axios.post('https://dummyjson.com/users/add', {
    firstName: 'Mahdi',
    lastName: 'Ansari',
    age: 30,
    /* other user data */
  })
  .then(function (response) {
    console.log(response);
  });
```

2.3 Setup for the Lab

2.3.1 Initiating the React app

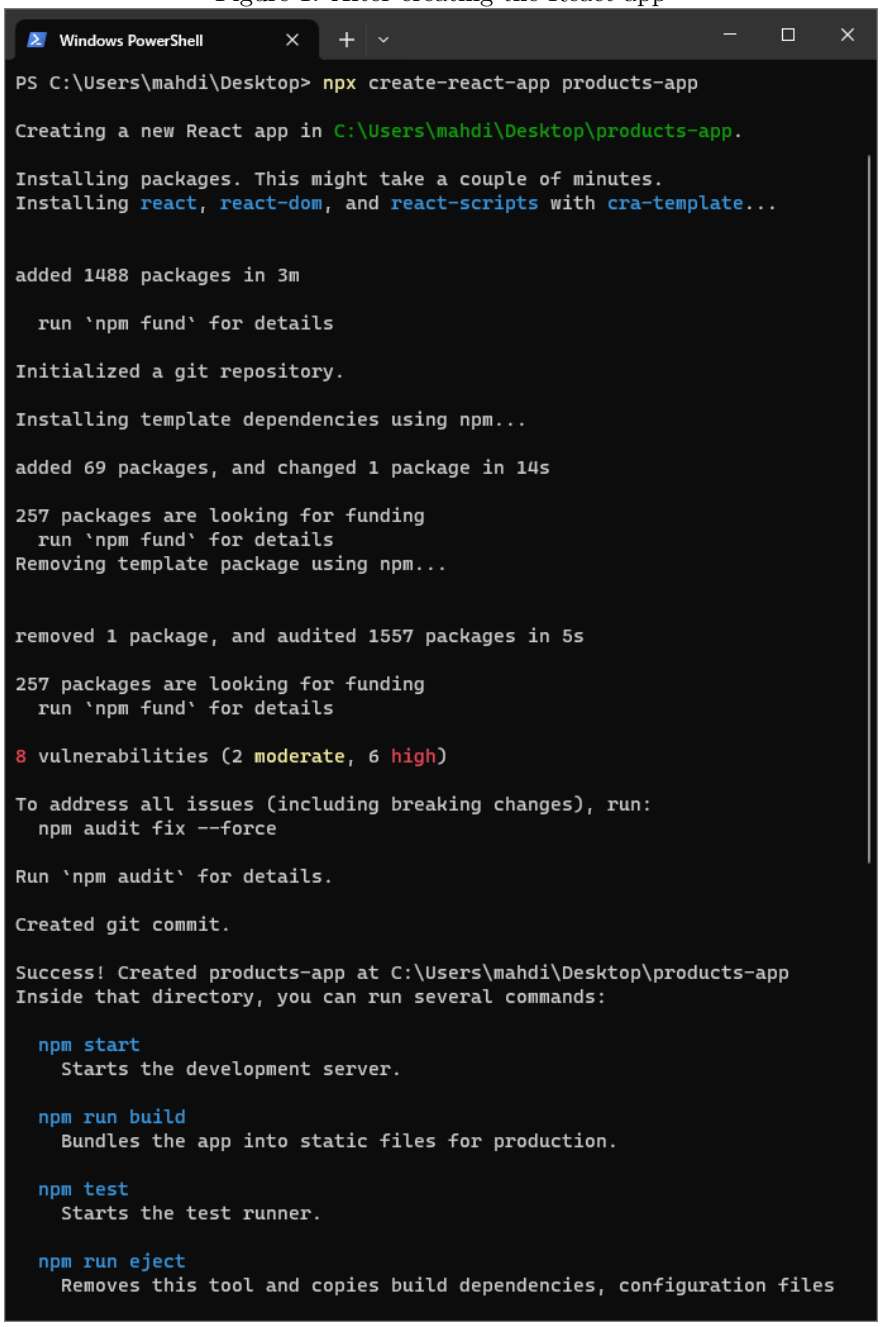
Here are the steps to follow:

1. Make sure you have a recent version of Node.js installed.
2. We normally use the following command to make a new react project. Let's create our project by using this command.

```
npx create-react-app products-app
```

3. Run the original React app using '`npm start`'. It normally opens a new tab in your browser using the `http://localhost:3000/` address. It should display the famous React start page.

Figure 1: After creating the React app




```
cd ~/lab8
npm start
```

4. After confirming the React app works, stop the development server by using '`Ctrl + C`' twice.
5. If you noticed during the creation of the React app, a git repository is also initiated, as indicated by the presence of a '.git' folder in the application folder. Also, as shown in Figure 1 a log says, 'Created git commit.'
6. Create an empty repository on GitHub and name it 'Lab8'.

7. Use your preferred git client application and set the remote repository of the local git repository using the URL of the repository you recently created. You must already know how to use git client applications, so we do not repeat instructions here. However, if you are a `SourceTree` lover like me, you must use the 'Add' option first to add an existing local repository to your git client, then set its remote or origin, and finally `push` your current commits.

8. Only push the initial state to the remote repository. At this point, your remote repository must have only one commit with the auto-generated comment 'Initialize project using the Create React App'.

9. Do not forget to invite your colleague to your repository. Any individual submission will not be evaluated.

2.4 Deliverable

1. Add the URL of your GitHub repository to your answer sheet. Ensure the repository is public and remains undeleted until the semester's end.

3 Exercise B (Listing Products)

3.1 Installing Necessary Packages

1. You need ensure all your project dependencies are up to date. This is crucial for compatibility with Node.js 20 and OpenSSL 3.0. Update your package.json or directly run the following commands.

2. The create-react-app toolchain (via react-scripts) should also be up-to-date to ensure compatibility with your Node.js environment. Update react-scripts to the latest version also.

```
npm update
npm install react-scripts@latest
```

3. You will need to install latest `axios` for making HTTP requests more conveniently.

```
npm install axios@latest
```

Note:

If you encounter, **ERR_OSSL_EVP_UNSUPPORTED** error later when you run the application, it is related to Node.js 17 and later versions which use OpenSSL 3.0. This version of OpenSSL changes how algorithms are loaded and can cause issues with certain Node.js operations, particularly when working with cryptographic functions that might be indirectly used by various dependencies in your project (i.e., here HTTPS communication in axios). Make sure you have run the above commands precisely.

4. You can use React Router for navigating between different views.

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

5. You can include Bootstrap in your React project for enhancing your design.

6. You can include FontAwesome in your project to add beautiful icons in your design. Run the following commands to add them to your project.

```
npm install bootstrap
npm install --save @fortawesome/react-fontawesome @fortawesome/free-solid-svg-
    icons @fortawesome/fontawesome-svg-core
```

3.2 Setting Up the Application Structure

Create a folder structure within the `src` directory for organizing your components, context, and services:

- `components/` will hold all your React components.
- `context/` will be used for global state management using `useContext`.
- `services/` will contain functions for API calls.

We strongly recommend typing code by hand as you are working through the lab and not using copy and paste. This will help you develop muscle memory and a stronger understanding. However, if insist on copy-pasting, you can find the code of each listing in a different text file.

3.3 Implementing the API Service

1. **Create `apiService.js` in `services` folder:** Define functions for fetching products, searching products, getting product details, adding, editing, and deleting products using axios.

Listing 1: Content of src/services/apiService.js file

```
import axios from 'axios';

const BASE_URL = 'https://dummyjson.com/products';

// Fetches a list of all products
export const fetchProducts = async () => {
    try {
        const response = await axios.get(BASE_URL);
        return response.data;
    } catch (error) {
        throw error; // Rethrow to allow caller to handle
    }
};

// Fetches details for a single product by its ID
export const getProductDetails = async (id) => {
    // implement it using similar logic as fetchProducts function
};

// Deletes a product by its ID
export const removeProduct = async (id) => {
    try {
        const response = await axios.delete(`${BASE_URL}/${id}`);
        return response.data;
    } catch (error) {
        throw error; // Rethrow to allow caller to handle
    }
};

// Adds a new product to the database
export const addProduct = (product) => {
    return axios.post(`${BASE_URL}/add`, JSON.stringify(product));
};

// Edits an existing product by ID
export const editProduct = (id, product) => {
    // implement it using similar logic as addProduct function but use axios.
       put method
};
```

3.4 Creating Context for State Management

1. **Create `ProductsContext.js` in `context` folder:** One of the best ways to provide functions or state across different components in your application is through React's Context API. This is especially useful if you need some event handlers like `onProductSaved` to be accessible in various parts of your app.

   Create a `ProductContext.js` file to hold your callback function and provide it at a high level in your app. Use `createContext` and `useContext` to manage and provide global state for your products.

Listing 2: Content of src/context/ProductsContext.js file

```
import React, { createContext, useContext, useState } from 'react';

const ProductsContext = createContext();

export const useProductsContext = () => useContext(ProductsContext);

export const ProductsProvider = ({ children }) => {
  const [products, setProducts] = useState([]);

  const saveProduct = (product) => {
    setProducts(currentProducts => {
      // Check if the product with the given ID already exists
      const index = currentProducts.findIndex(p => p.id === product.id);

      if (index !== -1) {
        // Product exists, replace it
        const newProducts = [...currentProducts];
        newProducts[index] = product;
        return newProducts;
      } else {
        // Product doesn't exist, add it as a new product
        // If the product doesn't have an ID, generate one
        const newProduct = product.id ? product : { ...product, id: Date.now()
            };
        return [...currentProducts, newProduct];
      }
    });
  };

  return (
    <ProductsContext.Provider value={{ products, setProducts, saveProduct }}>
      {children}
    </ProductsContext.Provider>
  );
};
```

3.5 Building the React Components

In this section we are going to define some components that we will use them later in *App.js* file.

1. **Create** `HomePage.js`**:** This component will display basic information about our application. Don't forget to replace your team member's name in the table. If you forget to update the names, you will get a zero because it shows you just copied and pasted codes.

Listing 3: Content of src/components/HomePage.js file

```
import React from 'react';

// using JSON for styling
const styles = {
    container: {
        padding: '10px 1vw',
    },
    table: {
        marginLeft: '2vw',
        width: '100px',
        borderCollapse: 'collapse',
```

```
    },
    th: {
        borderBottom: '1px solid #ddd',
        padding: '8px',
        textAlign: 'left',
    },
    td: {
        borderBottom: '1px solid #ddd',
        padding: '8px',
        textAlign: 'left',
    }
};

const HomePage = () => {
    return (
        <div style={styles.container}>
            <h2>Welcome to Our Project</h2>
            <p>This project is designed to showcase our abilities to work with
                React, including state management, routing, and interacting
               with an API.</p>

            <h3>Creators:</h3>
            <table style={styles.table}>
                <thead>
                    <tr>
                        <th style={styles.th}>Name</th>
                        <th style={styles.th}>Role</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td style={styles.td}>A B</td>
                        <td style={styles.td}>Developer</td>
                    </tr>
                    <tr>
                        <td style={styles.td}>C D</td>
                        <td style={styles.td}>Designer</td>
                    </tr>
                </tbody>
            </table>
        </div>
    );
};

export default HomePage;
```

Note:

Using a JSON object for styling in JavaScript applications, often seen in React projects, involves defining CSS styles within a JavaScript object, as shown in the `styles` object in above example. This approach, known as inline styling when applied via the style attribute, has several pros and cons. You have to investigate about their pros and cons and reflect it in your answer sheet file.

2. **Create** `NotFoundPage.js`**:** To handle undefined or error routes in React Router v6, you can use a Route with the path set to '*'. This will match any URL that doesn't match the other specified routes. You can create a component to represent a 404 Not Found page or any other error message and render it for this catch-all route. First, we must define an error component, for example, `NotFoundPage.js`.

Listing 4: Content of src/components/NotFoundPage.js file

```
import React from 'react';

const NotFoundPage = () => {
  return (
    <div>
      <h2>404 Not Found</h2>
      <p>The page you're looking for does not exist.</p>
    </div>
  );
};

export default NotFoundPage;
```

3. **Create** `ProductList.js`**:** This component will display a list of products. Use `useEffect` to fetch products on component mount and `useState` to manage the products state locally.

Listing 5: Content of src/components/ProductList.js file

```
import React, { useEffect, useState } from 'react';
import { fetchProducts } from '../services/apiService';
import { useProductsContext } from '../context/ProductsContext';

const ProductList = () => {
  const { products, setProducts } = useProductsContext();

  useEffect(() => {
    const getProducts = async () => {
      const data = await fetchProducts();
      setProducts(data.products);
    };
    getProducts();
  }, [setProducts]);

  return (
    <div>
      {products.map((product) => (
        <div key={product.id}>{product.title}</div>
      ))}
    </div>
  );
};

export default ProductList;
```

3.6 Implementing Navigation

You can use React Router for navigating between different views. Set up routes in `App.js` for navigating between your components.

Listing 6: Content of src/App.js file

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom
    ';
import { ProductsProvider } from './context/ProductsContext'; // Adjust the
    import path as necessary
```

```
import ProductList from './components/ProductList';
import NotFoundPage from './components/NotFoundPage';
import HomePage from './components/HomePage';
import './App.css';

function App() {
  return (
    <div className="App">
      <ProductsProvider>
        <Router>
          <div>
            {/* Navigation Links */}
            <nav>
              <ul>
                <li>
                  <Link to="/">Home</Link>
                </li>
                <li>
                  <Link to="/products">Product List</Link>
                </li>
                <li>
                  <Link to="/product/add">Add Product</Link>
                </li>
              </ul>
            </nav>

            {/* Routes for different pages */}
            <Routes>
              <Route path="/products" element={<ProductList />} />
              <Route path="/" element={<HomePage />} />
              <Route path="*" element={<NotFoundPage />} /> {/* Catch all
                  other route */}
            </Routes>
          </div>
        </Router>
      </ProductsProvider>
    </div>
  );
}

export default App;
```

Let's update the `App.css` file too:

Listing 7: Content of src/App.css file

```
.App {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: start;
  justify-content: start;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

3.7 Running the Application

1. **Start Your Application:**

```
npm start
```

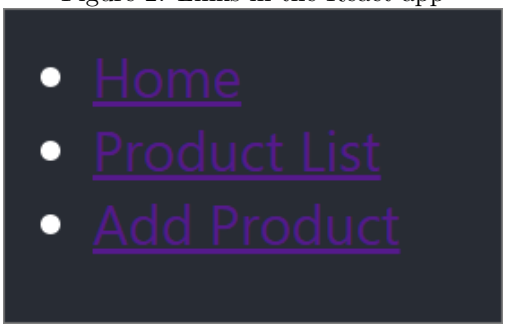
2. **Visit `http://localhost:3000` to view your application.**

3.8 Deliverable

1. First, format all the edited or created files with the VSC formatter, then commit and push your final code to the repository using 'Step B' as comment of your commit. Make sure you have done all the steps exactly as we explained in this instruction.

2. In the app you have three links (i.e., Home, Product List, Add Product). Press on each link and make a full screen screenshot from each page. The screenshots must include the address bar also.

Figure 2: Links in the React app



3. Add the screenshot of the 'Home' page, to your answer sheet file.

4. Add the screenshot of the 'Product List' page, to your answer sheet file.

5. Add the screenshot of the 'Add Product' page, to your answer sheet file. Why do you see 'Not found' page when pressing on 'Add Product' link?

6. Investigate and discuss the pros and cons of using a JSON object for styling in React applications, and list two pros and two cons in your answer sheet. Please do not use ChatGPT. Remember to include your references. Note that search engines, ChatGPT, or websites like Wikipedia, which aggregate information from various sources, cannot be used as references in an academic environment.

4 Exercise C (Complete Products App Functionalities)

4.1 Building Rest of the React Components

In this section we are going to define some more components that we will use them later in *App.js* file.

1. **Create `ConfirmModal.js` component:** For deleting a product we need to get a confirmation to make sure users do not delete items by mistake. We normally use `window.confirm` like this:

```
if (window.confirm('Are you sure you want to delete this record?')) {
   // Do some action
}
```

   Bootstrap provides a visually appealing and customizable modal component that you can use instead of the native `window.confirm`. This modal can serve as a dialog box for confirming actions like deleting a product. Using a Bootstrap modal instead of `window.confirm` allows for a better user experience, as you can style the modal to match your application's theme and provide more information or options to the user.

   Implementing a Bootstrap Modal for Confirmation: To use a Bootstrap modal as a confirmation dialog, you'll need to integrate the modal component into your React component. Here's a basic example of how you can implement a Bootstrap confirmation modal in a React component. This example assumes you're using Bootstrap 5 as we already installed it. First, create a reusable modal component. You can customize this component as needed, but here's a simple starting point:

Listing 8: Content of src/components/ConfirmModal.js file

```
import React from 'react';

const ConfirmModal = ({ show, onClose, onConfirm, title, children }) => {
  if (!show) {
    return null;
  }

  return (
    <div className="modal show d-block" tabIndex="-1" role="dialog">
      <div className="modal-dialog" role="document">
        <div className="modal-content">
          <div className="modal-header text-warning">
            <h4 className="modal-title">{title}</h4>
            <button type="button" className="btn-close" data-bs-dismiss="modal
                " aria-label="Close" onClick={onClose}></button>
          </div>
          <div className="modal-body text-secondary">{children}</div>
          <div className="modal-footer">
            <button type="button" className="btn btn-secondary" onClick={
                onClose}>
              Cancel
            </button>
            <button type="button" className="btn btn-danger" onClick={
                onConfirm}>
              Delete
            </button>
          </div>
        </div>
      </div>
    </div>
  );
```

```
};

export default ConfirmModal;
```

Using the Modal in Your Components: In your component where you handle the delete action, you would manage the state to control the visibility of the confirmation modal and handle the confirm action.

```
import React, { useState } from 'react';
import ConfirmModal from './ConfirmModal'; // Adjust the import path as
    necessary

const YourComponent = () => {
  const [showConfirmModal, setShowConfirmModal] = useState(false);

  const handleDelete = () => {
    // Logic to delete the item goes here
    setShowConfirmModal(false); // Close modal after confirmation
  };

  return (
    <>
      <button onClick={() => setShowConfirmModal(true)}>Delete Item</button>
      <ConfirmModal
        show={showConfirmModal}
        onClose={() => setShowConfirmModal(false)}
        onConfirm={handleDelete}
        title="Confirm Delete"
      >
        Are you sure you want to delete this item?
      </ConfirmModal>
    </>
  );
};
```

2. **Create `DeleteButton.js` component:** This component handles deleting a product. We'll implement a button that, when clicked, sends a delete request to the DummyJSON API to remove a specific product. After successfully deleting the product, it should trigger some action in the parent component, such as refreshing the list of products or showing a notification. It must use `removeProduct` function of services for the `onClick` event handler.

   Here's how to create the `DeleteButton.js` file. Navigate to your project directory, specifically within the `components` folder. Create a new file named `DeleteButton.js`. Add the following code to `DeleteButton.js` file:

Listing 9: Content of src/components/DeleteButton.js file

```
import React, { useState } from 'react';
import { removeProduct } from '../services/apiService';
import ConfirmModal from './ConfirmModal';

const DeleteButton = ({ productId, onProductDeleted }) => {
  const [showConfirmModal, setShowConfirmModal] = useState(false);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

```

```
  const handleDelete = async () => {
    try {
      setLoading(true);
      console.log(`Deleting product with id: ${productId}`);
      const data = await removeProduct(productId);
      setLoading(false);
      console.log(data);
      if (onProductDeleted) {
        onProductDeleted(data);
      }
    } catch (err) {
      console.error('Failed to delete product:', err);
      setError('Failed to delete product');
      setLoading(false);
    } finally {
      setShowConfirmModal(false); // Close modal after confirmation
    }
  };

  return (
    <>
      <button className="btn btn-danger" onClick={() => setShowConfirmModal(
          true)} disabled={loading}>
        {loading ? 'Deleting...' : 'Delete Product'}
      </button>
      {error && <p>{error}</p>}
      <ConfirmModal
        show={showConfirmModal}
        onClose={() => setShowConfirmModal(false)}
        onConfirm={handleDelete}
        title="Confirm Delete"
      >
        Are you sure you want to delete this product?
      </ConfirmModal>
    </>
  );
};

export default DeleteButton;
```

This code does the following:

- Defines a `DeleteButton` component that takes a `productId` and an `onProductDeleted` callback as props.
- Uses `useState` to manage the loading and error states.
- Defines a `handleDelete` function that shows a confirmation dialog to the user. If the user confirms, it sends a delete request to the API.
- On successful deletion, it calls the `onProductDeleted` callback, passing the `productId` so the parent component can react accordingly (e.g., by removing the product from the state, thus updating the UI).
- Displays a loading state in the button text while the delete operation is in progress and shows any error message if the operation fails.

Integration with Parent Component: To use the `DeleteButton` component effectively, you need to handle the `onProductDeleted` callback in your parent component:

- **Add the `DeleteButton` component** where you list or display your products, passing the required `productId` and a handler for `onProductDeleted`.
- **Implement the `onProductDeleted` handler** in the parent component to update the UI after a product is deleted. This could involve filtering out the deleted product from your products state or re-fetching the products list from the API.

Example in a parent component:

```
const handleProductDeleted = (deletedProductId) => {
  // Update your state to reflect the deletion
  // For example, if you have a state variable `products`
  // holding your list of products:
  setProducts(currentProducts =>
  currentProducts.filter(product => product.id !== deletedProductId));
  // Or show a success message, or navigate away, etc.
  alert('Product deleted successfully!');
};
```

And then include the `DeleteButton` in your product list or detail view:

```
<DeleteButton productId={product.id}
   onProductDeleted={handleProductDeleted} />
```

This setup allows you to integrate product deletion functionality into your React application seamlessly, with feedback to the user and immediate updates to the UI.

3. **Create `ProductDetails.js` file:** This component will show details of a selected product. To create the `ProductDetails.js` component, we will fetch and display the details of a selected product from the DummyJSON API. It uses `getProductDetails` function of the services to fetch data. This component will use the `useEffect` hook to fetch product details when the component mounts or when the selected product ID changes, and the `useState` hook to store the product's details.

   Here's how you can create the `ProductDetails.js` file. Navigate to your project directory (if you haven't already) and open it in your preferred code editor. Create a new file in the components directory named `ProductDetails.js`. Add the following code to `ProductDetails.js`:

Listing 10: Content of src/components/ProductDetails.js file

```
import React, { useState, useEffect } from 'react';
import DeleteButton from './DeleteButton';
import { getProductDetails } from '../services/apiService';
import { useParams } from 'react-router-dom';
import { useNavigate } from 'react-router-dom';

const ProductDetails = () => {
  // Access route parameter
  const { productId } = useParams();
  const navigate = useNavigate();

  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchProductDetails = async () => {
      setLoading(true);
      try {
        const data = await getProductDetails(productId);
        setProduct(data);
```

```
        setError('');
      } catch (error) {
        setError('Failed to fetch product details.');
        console.error(error);
      } finally {
        setLoading(false);
      }
    };
    if (productId) {
      fetchProductDetails();
    }
  }, [productId]);

  // Function to handle after delete
  const onProductDeleted = async (product) => {
    navigate(-1);
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;
  if (!product) return <div>No product found.</div>;

  return (
    <div className="container mt-4">
      <div className="row">
        <div className="col-md-4 d-flex justify-content-center align-items-
            start">
          <img src={product.thumbnail} alt={product.title} className="img-
              fluid" />
        </div>
        <div className="col-md-8">
          <h2>{product.title}</h2>
          <p>{product.description}</p>
          <ul className="list-unstyled">
            <li>Price: ${product.price}</li>
            <li>Category: {product.category}</li>
            <li>Brand: {product.brand}</li>
          </ul>
          <button className="btn btn-primary me-2" onClick={() => navigate(`/
              product/edit/${product.id}`)}>Edit Product</button>
          <DeleteButton productId={product.id} onProductDeleted={
              onProductDeleted} />
        </div>
      </div>
    </div>
  );
};

export default ProductDetails;
```

This code does the following:

- It defines a `ProductDetails` component that takes the `productId` from the URL parameters.
- It uses `useState` to manage the product's details, loading state, and any potential error message.
- It uses `useEffect` to make an HTTP request to the DummyJSON API to fetch the details of a product when the component mounts or the `productId` changes.

- It renders the product's details, including the title, description, thumbnail, price, category, and brand. If the product is loading, it displays a loading message. If there's an error, it displays an error message.
- It consumes `DeleteButton` component and pass the `onProductDeleted` as a call back function.

Note:

Please notice that we used Bootstrap classes for decorating HTML elements.

4. **Update `ProductList.js` component:** Now that we defined a component for showing details of a product, we can make each row in our product list clickable and navigate to the product details page upon click. We can use React Router's `useNavigate` hook for functional components. This approach allows us to programmatically navigate to the details page for each product.

   Using Bootstrap's List Group with Clickable Rows: Here, we make each list group item clickable. When clicked, the user is navigated to the product details page for the clicked product.

   Adding Custom Hover Highlight to List or Table Rows: We can add a custom CSS for hover effects on list items. We will add a highlight effect to rows in a Bootstrap table or list group items by using a custom CSS class `hover-highlight`. We will later provide its CSS definition in `App.css` file.

Listing 11: Content of src/components/ProductList.js file

```
import React, { useEffect, useState } from 'react';
import { fetchProducts } from '../services/apiService';
import { useProductsContext } from '../context/ProductsContext';
import { useNavigate } from 'react-router-dom';

const ProductList = () => {
  const { products, setProducts } = useProductsContext();
  const navigate = useNavigate();

  useEffect(() => {
    const getProducts = async () => {
      const data = await fetchProducts();
      setProducts(data.products);
    };
    getProducts();
  }, [setProducts]);

  return (
    <div className="list-group">
      {products.map((product, index) => (
        <button
          key={product.id}
          className="list-group-item d-flex justify-content-between align-
              items-center hover-highlight"
          onClick={() => navigate(`/product/${product.id}`)}
        >
          <span>{index + 1}. {product.title}</span>
        </button>
      ))}
    </div>
  );
};

export default ProductList;
```

In this example, each product row is rendered as a `button` with `list-group-item` styling. The `onClick` event handler uses `navigate` to redirect the user to the product's detail page.

5. **Create `ProductForm.js` file:** We need a form for adding or editing products. To create the `ProductForm.js` component for adding and editing product details, we'll implement a form that submits product data to the DummyJSON API. It must use `addProduct` and `editProduct` methods from your service. This component will handle both creating new products and updating existing ones, depending on whether a product ID is passed as a prop.

   Here's how you can create the `ProductForm.js` file. Navigate to your project directory and ensure you're in the components folder. Create a new file named `ProductForm.js`. Add the following code to `ProductForm.js`.

Listing 12: Content of src/components/ProductForm.js file

```
import React, { useState, useEffect } from 'react';
import { addProduct, editProduct, getProductDetails } from '../services/
    apiService';
import { useProductsContext } from '../context/ProductsContext';
import { useParams } from 'react-router-dom';
import { useNavigate } from 'react-router-dom';

const ProductForm = () => {
  // Access route parameter
  const { productId } = useParams();
  const navigate = useNavigate();

  const { saveProduct } = useProductsContext();
  const [product, setProduct] = useState({
    title: '',
    description: '',
    price: '',
    brand: '',
    category: '',
  });
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchProductDetails = async () => {
      setLoading(true);
      try {
        const data = await getProductDetails(productId);
        setProduct(data);
        setError('');
      } catch (error) {
        console.error("Error fetching product details:", error);
        setLoading(false);
        setError('Failed to load product details');
      } finally {
        setLoading(false);
      }
    };
    if (productId) {
      fetchProductDetails();
    }
  }, [productId]);

  const handleChange = (e) => {
```

```
    const { name, value } = e.target;
    setProduct(prevState => ({
      ...prevState,
      [name]: value,
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);

    const apiCall = productId ? editProduct(productId, product) : addProduct(
        product);

    apiCall.then(response => {
      setLoading(false);
      saveProduct(response.data);
      // Navigate back
      navigate(-1);
    })
    .catch(error => {
      console.error("Error saving product:", error);
      setError('Failed to save product');
      setLoading(false);
    });
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label htmlFor="title" className="form-label">Title:</label>
        <input type="text" className="form-control" id="title" name="title"
            value={product.title} onChange={handleChange} required />
      </div>

      <div className="mb-3">
        <label htmlFor="description" className="form-label">Description:</
            label>
        <textarea className="form-control" id="description" name="description"
             rows="3" value={product.description} onChange={handleChange}
            required></textarea>
      </div>

      <div className="mb-3">
        <label htmlFor="price" className="form-label">Price:</label>
        <input type="number" className="form-control" id="price" name="price"
            value={product.price} onChange={handleChange} required />
      </div>

      <div className="mb-3">
        <label htmlFor="brand" className="form-label">Brand:</label>
        <input type="text" className="form-control" id="brand" name="brand"
            value={product.brand} onChange={handleChange} />
      </div>

      <div className="mb-3">
```

```
        <label htmlFor="category" className="form-label">Category:</label>
        <input type="text" className="form-control" id="category" name="
            category" value={product.category} onChange={handleChange} />
      </div>

      <button type="submit" className="btn btn-primary">Save Product</button>
    </form>
  );
};

export default ProductForm;
```

This code does the following:

- Defines a `ProductForm` component that optionally gets the `productId` from URL parameters and a callback function `saveProduct` from context.
- Uses `useState` to manage the form fields (as part of the `product` object) and loading/error states.
- On component mount, if a `productId` is provided, it fetches the product details from the API and pre-populates the form fields, allowing for edit functionality.
- Handles form field changes to update the component's state.
- On form submission, it determines whether to create a new product or update an existing one based on the presence of `productId` and sends the appropriate request to the API.
- Calls the `saveProduct` callback function upon successful save, allowing parent components to react to the save operation (e.g., refreshing the product list or navigating to a different view). Please check the context to see how we defined the `saveProduct` function.

4.2 Final Integration and Testing

1. **Integrate all components in `App.js`:** Let's change our `App.js` to have a header navbar using icons, and supporting rest of the routes.

Listing 13: Content of src/App.js file

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom
    ';
import { ProductsProvider } from './context/ProductsContext'; // Adjust the
    import path as necessary
import ProductList from './components/ProductList';
import ProductDetails from './components/ProductDetails';
import ProductForm from './components/ProductForm';
import NotFoundPage from './components/NotFoundPage';
import HomePage from './components/HomePage';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faHome, faList, faPlus } from '@fortawesome/free-solid-svg-icons';
import 'bootstrap/dist/css/bootstrap.min.css';
import './App.css';

function App() {
  return (
    <div className="App">
      <ProductsProvider>
        <Router>
          <div>
```

```
            {/* Navigation Links */}
            <nav className="navbar navbar-expand navbar-light full-width">
              <div className="container-fluid">
                <div className="collapse navbar-collapse" id="navbarNav">
                  <ul className="navbar-nav nav-full-width">
                    <li className="nav-item">
                      <Link className="nav-link" to="/" data-bs-toggle="
                          tooltip" data-bs-placement="bottom" title="Home">
                        <FontAwesomeIcon icon={faHome} />
                      </Link>
                    </li>
                    <li className="nav-item">
                      <Link className="nav-link" to="/products" data-bs-toggle
                          ="tooltip" data-bs-placement="bottom" title="Product
                          List">
                        <FontAwesomeIcon icon={faList} />
                      </Link>
                    </li>
                    <li className="nav-item">
                      <Link className="nav-link" to="/product/add" data-bs-
                          toggle="tooltip" data-bs-placement="bottom" title="
                          Add Product">
                        <FontAwesomeIcon icon={faPlus} />
                      </Link>
                    </li>
                  </ul>
                </div>
              </div>
            </nav>


            {/* Routes for different pages */}
            <div className="container mt-3">
              <Routes>
                <Route path="/products" element={<ProductList />} />
                <Route path="/product/:productId" element={<ProductDetails />}
                     />
                <Route path="/product/add" element={<ProductForm />} />
                <Route path="/product/edit/:productId" element={<ProductForm
                    />} />
                <Route path="/" element={<HomePage />} />
                <Route path="*" element={<NotFoundPage />} />
              </Routes>
            </div>
          </div>
        </Router>
      </ProductsProvider>
    </div>
  );
}

export default App;
```

2. **Change** `App.css`**:** We need to update our `App.css` file also to support our navbar and highlighting of list items.

Listing 14: Content of src/App.css file

```css
.App {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: start;
  justify-content: start;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App > :first-child {
  width: 100%;
  align-self: stretch;
}

.navbar {
  background-color: #333740;
}

.nav-link {
  color: #ffffff;
  transition: color 0.2s;
}

.nav-link:hover, .nav-link:focus {
  color: #cccccc;
  text-decoration: none;
}

.nav-full-width {
  width: 100%;
  justify-content: space-between;
}

/* Tooltip styling */
.tooltip-inner {
  background-color: #f0f0f0;
  color: #282c34;
  padding: 5px 10px;
  border-radius: 4px;
  font-size: 0.875rem;
}

/* for hovering list's items */
.hover-highlight:hover {
  background-color: #f8f9fa !important;
}
```

3. **Test Your Application:** Ensure that all functionalities (listing, searching, viewing details, adding, editing, and removing products) work as expected.

4.3 Deliverable

1. First, format all the edited or created files with the VSC formatter, then commit and push your final code to the repository using 'Step C' as comment of your commit. Make sure you have done all the steps exactly as we explained in this instruction.

2. In the app you have four pages (i.e., Home, Product List, Product Details, Edit Product). Navigate to each page and make a full screen screenshot from each page based on the following conditions. The screenshots must include the address bar also.
3. Add the screenshot of the 'Home' page, to your answer sheet file.
4. Add the screenshot of the 'Product List' page, to your answer sheet file.
5. Add the screenshot of the 'Product Details' page of 'Huawei P30' product, to your answer sheet file. Any other product does not have any mark!
6. Add the screenshot of the 'Edit Product' page of 'iPhone x' product, to your answer sheet file. Any other product does not have any mark!

Note:

We will use this code later for testing our own back-end in future labs.