HyperionDev

React – Routing

Task

Visit our website

Introduction

Welcome to the React – Routing task! In this task, you will learn how to implement navigation within a React application using React Router. React is commonly used to build single-page applications (SPAs), which are web apps that load a single **index.html** file and dynamically update content without requiring full-page reloads. Unlike traditional websites, which reload the entire page for each navigation, SPAs provide a faster, more seamless user experience by updating the view as users interact with the app. React Router enables you to map URLs to specific React components, allowing the app to change its displayed content based on the URL while maintaining the benefits of an SPA. Before starting this task, make sure you're comfortable with React components, state, functions, and conditional rendering.

Take note

Before taking on this task, it's crucial that you have a solid understanding of these topics:

- React components
- React state
- Functions and parameters
- Conditional rendering

React router

The primary purpose of React Router is to enable intuitive navigation within a React application. React Router allows you to build navigation menus and links, and also handle the rendering of components based on the specified routes. It's a crucial tool for creating SPAs where the content dynamically changes without a full-page reload. This way, users can access different sections or views of the application without having to load entirely new pages, improving the overall user experience.

Take note

You are about to encounter **16 steps** that explain the process of setting up and implementing routing in a React application. These steps will take you from setting up your project to creating and connecting components, ensuring a seamless navigation experience within your app. The steps are found across the different sections of this task.

To use `react-router`, do the following:

Step 1: Navigate to your React application.

Create a new React application using the **React + Vite starter kit** by following the **setup instructions**, and then navigate to the directory of your new application to continue with the following code:

```
cd <react-project-directory>
```

Step 2: Install the necessary dependencies by running:

```
npm install
```

Step 3: Install `react-router-dom`.

To add routing capabilities to your application, install React Router by running the following command:

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

Running this command will install the latest version of `react-router-dom`, providing the components and functions needed for setting up routes in your React app.

Step 4: Set up routes using `createBrowserRouter`.

In your project's main file (typically **src/main.jsx**), create and render a browser router using React Router's latest syntax:

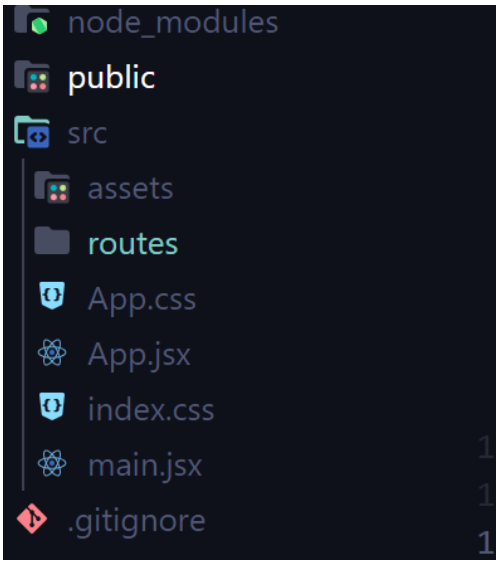
```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import "./index.css";

const router = createBrowserRouter([
  {
    path: "/",
    element: <div>Hello world!</div>,
  },
]);
```

```
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

The first route in our configuration, often called the “root route,” serves as the foundation for other routes in the application. For example, the root route may use the path “/” as the main entry point of the app. This route defines the primary layout of the user interface, with other routes rendering inside it as users navigate through different sections. By establishing a root route, we create a central starting point for navigation. Other routes can then display different components or views within the same layout, allowing the application to feel cohesive as users navigate without needing a full-page reload.

Step 5: Create the navigation for the application.

Open **src/** and create a **routes** folder:



Step 6: Inside the **routes** folder create a component named **NavBar.jsx**. This component is used to show a list of navigational links that will be positioned at the top of the application.

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

export default function NavBar() {
  return (
    <div className="navbar">
      <nav>
        <ul className="nav-links">
```

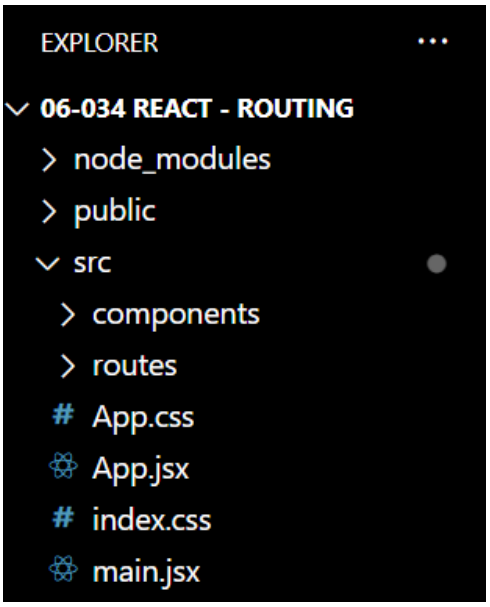
```
          <li>
            <Link to="/" className="nav_link">
              Home
            </Link>
          </li>
          <li>
            <Link to="/products" className="nav_link">
              Products
            </Link>
          </li>
          <li>
            <Link to="/contacts" className="nav_link">
              Contacts
            </Link>
          </li>
        </ul>
      </nav>
    </div>
  );
}
```

- **Line 1:** imports the `Link` component from the `react-router-dom` library.
- **Line 9:** link to navigate to the `Home` component.
- **Line 14:** link to navigate to the `Products` component.
- **Line 19**: link to navigate to the `Contacts` component.

The `Link` component provides declarative, accessible navigation around your application. For example, `<Link to="/">Home</Link>` will take the user to the `Home` component when "Home" is clicked on. Note that the `Link` component is used to navigate to a particular route. This is similar to using the anchor tag (`<a>`) in HTML. The `to="/"` will contain the path name defined in the route you wish to navigate to.

Step 7: Create components for the application.

Open **src/** and create a **components** folder:



Below you'll see how to create three individual component files. Create these files and place them inside your **components** folder. This will allow you to create a better file structure that will make your code easier to read and maintain. For more information click **here**.

Step 8: Create the **Home**, **Products**, and **Contacts** components.

The following code creates the **Home.jsx** component:

```
import NavBar from "../routes/NavBar";   //import the navigation bar component

export default function Home() {
  return (
    <div>
       {/* Display the navigation bar at the top of the page */}
        <NavBar />
        <h1>Welcome to the homepage!</h1>
    </div>
  );
}
```

The following code creates the **Products.jsx** component:

```
import NavBar from "../routes/NavBar"; //import the navigation bar component

export default function Products() {
  return (
    <div>
       {/* Display the navigation bar at the top of the page */}
        <NavBar />
        <h1>Products Page</h1>
```

```
    </div>
  );
}
```

The following code creates the **Contacts.jsx** component:

```
import NavBar from "../routes/NavBar"; //import the navigation bar component

export default function Contact() {
  return (
    <div>
        {/* Display the navigation bar at the top of the page */}
        <NavBar />
        <h1>Contact Page</h1>
    </div>
  );
}
```

Step 9: Import the created components in **main.jsx** and create new routes inside of the `createBrowserRouter` router:

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import "./index.css";

/* import the newly created components */
import Home from "./components/Home";
import Products from "./components/Products";
import Contacts from "./components/Contacts";

/*create a browser router that routes to the components*/
const router = createBrowserRouter([
  {
    path: "/",
    element: <Home />,
  },
  {
    path: "/products",
    element: <Products />,
  },
  {
    path: "/contacts",
    element: <Contacts />,
  },
]);

createRoot(document.getElementById("root")).render(
  <StrictMode>
```

```
    <RouterProvider router={router} />
  </StrictMode>
);
```

The `path=""` rules describe which components will be visible on the screen based on the URL path defined in the `route` element. For example, in the code above:

- `path="/contact" element=<Contact/>;` will display the `Contacts` component on the screen when the URL in your browser's address bar contains the string `"/contact"`, e.g., “http://localhost:5173/contact”.
- `exact path="/" element={<Home />;` will display the `Home` component as the root URL, e.g., http://hyperiondev.com/. The default root when running your React app on your local machine would be similar to “http://localhost:5173/”, which would be your root URL.

Extra resource

For more information about using React Router, read more in the **documentation**.

Keeping state values

React Router, by default, unmounts the previous component and mounts the new component corresponding to the new route. This means that the state of the previous component will be reset to its initial values when you navigate to a new page.

If you need some data to persist between different pages or routes, there are a few options. You could use React Router hooks such as `useContext`, a global state manager, or local storage.

React Router hooks

React Router hooks are functions that allow you to interact with the routing system in React components. They provide features like accessing routing parameters, navigating programmatically, and managing URL parameters, all without using higher-order components like `<Route>` and `<Link>`. These hooks also allow you to store state outside of routes, keeping it intact across page navigations and making it accessible to other components in your app.

You can pass data through the URL when navigating between pages with the following hooks.

useNavigate() hook

The `useNavigate` hook returns a function that allows you to navigate to other pages programmatically. This is a great tool to use to programmatically navigate to another page within a few seconds of receiving data. For example, when a user clicks a button that sends some data to an API, as soon as the response object has been received we can then call the `useNavigate` hook to take the user to a new page. At the same time, the `useNavigate` hook allows you to pass the response data to the page being navigated to. Passing data through the state in URLs is useful as it avoids the need for worrying about URL parameters for sensitive data or using global state management tools like Redux.

Step 10: Update the code in **Home.jsx** with the following:

```
/*existing imports */
import { useNavigate } from "react-router-dom";

export default function Home() {
  const nav = useNavigate();

  return (
    <div>
      <button
        onClick={() => nav(`/products`, { state: { product: "iPhone 16" } })}
      >
        Buy
      </button>
    </div>
  );
}
```

- **Line 2:** imports the `useNavigate` hook from the `react-router-dom` library.
- **Line 5:** initialises the `nav` constant by calling the `useNavigate` hook. This hook provides the ability to navigate to different routes within the application. It's essentially extracting the navigate function from the `useNavigate` hook.

When the button is clicked, the `onClick` event is triggered. It calls an arrow function, which in turn calls the `nav` function with two arguments:

- The **first argument** is the route to navigate to, in this case, `/products`. This means the user will be redirected to the `/products` page when clicking the button.

- The **second argument** is an object containing additional state information to pass along with the navigation. In this example, it includes a product property with the value “iPhone 16” (this could also be any other value or state).

This information could be accessed on the destination route, typically through the `useLocation` hook.

useLocation() hook

The `useLocation` hook allows you to access the location of the object used by the `react-router`. This object represents the current URL and it’s immutable, which means that it cannot be changed. When the URL changes, `useLocation` returns a new location object that has been updated.

Step 11: Include the following code inside of **Products.jsx**:

```
import NavBar from "../routes/NavBar";   //import the navigation bar
import { useLocation } from "react-router-dom";

  export default function Products() {
    const location = useLocation();
    const data = location.state;

    return (
    <div>
      <NavBar />
      <h1>Products page</h1>
      <h3>
        {data.product || "No products"}
      </h3>
    </div>
  );
}
```

As soon as this component is rendered, the following will happen:

- **Line 2:** imports the `useLocation` hook from `react-router-dom`.
- **Line 5:** the `useLocation` hook is called and stored in a constant variable named `location`.
- **Line 6:** the `state` object is initialised and stored in a constant variable named `data`.
- **Line 13:** if a product is available it displays “No products”.

State management alternatives

An alternative to React Router hooks would be to use the `useContext` hook. The `useContext` hook in React is like a "delivery system" for sharing data between components without needing to pass props down through multiple levels.

Normally, to share data between components, you pass data from a parent component to its child through props. But if you have a deeply nested tree of components, passing props down many levels can become messy. The `useContext` hook helps you avoid this by letting you share data directly between components without "prop drilling". This is very useful for themes, user authentication, and settings that need to be accessed by many components.

How it works:

- **Context provider:** Think of this as a "supplier" of information. You wrap a group of components with a provider, and that provider makes the data available to any component inside it.
- **Context consumer:** Any component inside the provider can "consume" the data using the `useContext` hook. This hook tells React, "I want to access the shared data."

For example:

- You create a "context" to hold the data.
- You provide that data at a high level (using a provider).
- Any component inside that provider can grab the data using `useContext`.

Let's say you want to share a theme (light or dark) across the application that we have been building.

Step 12: In your **src** directory, create a **ThemeContext.jsx** file to define and export your context.

Your **ThemeContext.jsx** should look like this:

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

export const ThemeContext = createContext('light');

// Provider component that wraps your app
export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('dark'); // Manage the theme state
```

```
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

- **Line 1:** import the `createContext` and `useState` hooks from “react”.
- **Line 3:** create a context called `ThemeContext` with the default value of “light”.
- **Line 4:** create a provider component called `ThemeProvider`. This component will pass data to all of the children components that it’s wrapped around.

Step 13: Next, wrap your app with the **ThemeProvider** in **main.jsx**:

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import "./index.css";

// Importing the components
import Home from "./components/Home";
import Products from "./components/Products";
import Contacts from "./components/Contacts";

// Import the ThemeProvider
import { ThemeProvider } from "./ThemeContext";

// Create a browser router
const router = createBrowserRouter([
  {
    path: "/",
    element: <Home />,
  },
  {
    path: "/products",
    element: <Products />,
  },
  {
    path: "/contacts",
    element: <Contacts />,
  },
]);

// Render the router wrapped in the ThemeProvider
createRoot(document.getElementById("root")).render(
  <StrictMode>
    {/* Wrap the router in ThemeProvider */}
```

```
    <ThemeProvider>
      <RouterProvider router={router} />
    </ThemeProvider>
  </StrictMode>
);
```

Step 14: In **Home.jsx**, use the `useContext` hook to access the `theme` value.

```
import { useNavigate } from "react-router-dom";
import { useContext } from "react";
import { ThemeContext } from "../ThemeContext"; // Import ThemeContext

export default function Home() {
  const nav = useNavigate();

  // Get the theme and setTheme from the ThemeContext
  const { theme, setTheme } = useContext(ThemeContext);

  return (
    <div className={`app ${theme}`}>
      <button
        onClick={() => nav(`/products`, { state: { product: "iPhone 16" } })}
      >
        Buy
      </button>
      <br></br>
      <h1>Current Theme: {theme}</h1>
      <button onClick={() => setTheme(theme === "dark" ? "light" : "dark")}>
        Toggle Theme
      </button>
    </div>
  );
}
```

- **Line 2:** import `useContext` from “react”.
- **Line 3**: import `ThemeContext`.
- **Line 5:** get and set the context value (`theme`).
- **Line 8:** use the `theme` value as a `className` for the `div` element.

When the button is clicked, the `onClick` event is triggered. It calls an arrow function, which will toggle between the values “light” and “dark”. This will change the `className` of the `div` element, which will change the styling of the page.

Step 15: Apply styles based on the theme.

In your **index.css**, you can define the following styles for the application:

```
.app {
  display: flex;
  align-items: center;
  justify-content: center;
  text-align: center;
  flex-direction: column;
  height: 100vh;
}

/*styling for the dark theme */
.app.dark {
  background-color: black;
  color: white;
  }

/*styling for the light theme */
.app.light {
  background-color: white;
  color: black;
  }

/*styling for the navigation menu*/
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #333;
  color: #fff;
  padding: 1rem;
}

.nav-links {
  list-style-type: none;
  display: flex;
  margin: 0;
  padding: 0;
}

.nav-links li {
  margin-right: 1rem;
}

.nav-links a {
  color: #fff;
  text-decoration: none;
}
```

Step 16: To have the selected theme be consistent throughout the entire application, update the code inside **Products.jsx** and **Contacts.jsx** with the following:

Products.jsx

```
import { useLocation } from "react-router-dom";
import { ThemeContext } from "../ThemeContext";
import NavBar from "../routes/NavBar"; //import the navigation bar

export default function Products() {
  // Get the location object
  const location = useLocation();

  // Get the data from the location object
  const data = location.state;

  // Get the theme from the ThemeContext
  const { theme } = useContext(ThemeContext);

  return (
    <div>
      <NavBar />
      <h1>Products page</h1>
      {/* Conditionally render product if it exists */}
      <h3>{data.product || "No products"}</h3>
    </div>
  );
}
```

Contacts.jsx

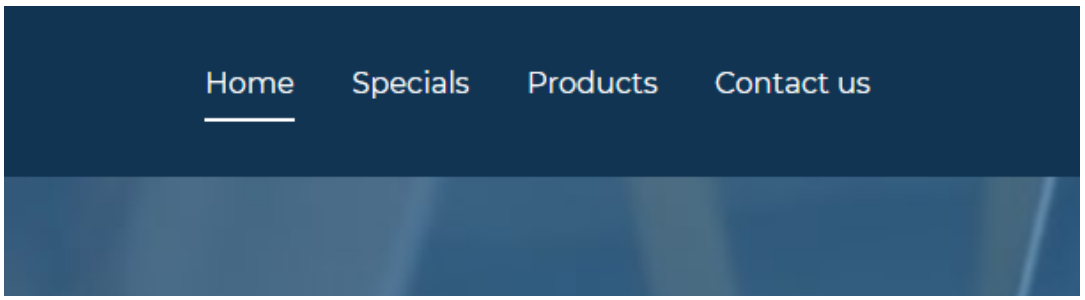
```
import { useContext } from "react";
import { ThemeContext } from "../ThemeContext";
import NavBar from "../routes/NavBar";

export default function Contact() {
  // Get the theme from the ThemeContext
  const { theme } = useContext(ThemeContext);

  return (
    <div>
        <NavBar />
        <h1>Contact Page</h1>
    </div>
  );
}
```

React Router library

The React Router library comes with many useful prebuilt hooks. For example, suppose you want to render content like an image or text based on the URL. In that case, the `useLocation` hook can also help you achieve this goal. Or if you want your application to

automatically navigate to another page after receiving API data or running some logic, the `useNavigate` hook can achieve this.

This is an example of how routes can be used to build a navigation menu in React:



Every link in this menu will display a different component and also change the URL.

Take note

To learn more about Router hooks and to build a more robust and intuitive application, check out the **source documentation**.

Practical task 1

You are going to create a fictional multi-page online store web application. This will be a three-page store with a navigation menu that allows users to switch between different pages. The navigation menu will look similar to the example image above in the “React Router library” section.

Follow these steps:

- Use `react-router` to create a navigation menu component that appears on every page of your application.
- This navigation menu should contain at least three links to the following components: `Home`, `Products`, and `About`. The “Home” link should be set as the root URL (“`/`”) of your application. When a user clicks on any of these links, only the corresponding component should be displayed.

- On the “Home” page, you need to create an input field and a “Login” button. The input field should request the user to enter their name.
- Once the user enters their name and clicks the “Login” button, the input field should be replaced with an `<h1>` tag saying, “Welcome (user name)”, and the “Login” button text should change to “Logout”.
- The user should not be able to log in if the input field is empty.
- If the “Logout” button is clicked, the welcome message should disappear, and the input field should reappear for the user to enter their name again. Then, the “Logout” button should change to the “Login” button again.

Be sure to place files for submission inside your task folder and click “Request review” on your dashboard.

Practical task 2

Follow these steps:

- In your `Products` component, display at least ten product items on the page using **this Bootstrap card**. Make use of an array of objects and render each product using the `.map()` method.
- Each product should have a responsive image, a title, a short description, a price, and a “Buy” button.
- Each product should also have at least three colour options. Use a Bootstrap **dropdown button** for this.
- When a user selects a colour from the dropdown menu, the text displayed on the button should update to show the name of the selected colour.
- Create a component called `TotalPrice.jsx` and define an `<h2>` tag that says “Total price: ”. This will be used to display the total price of both products.

- When the “Buy” button is clicked, the total price component should be updated to display the total price of all purchased products.
- The total price component should be imported into every other component **except** the `Home` component and displayed at the top-right corner. However, the total price component should only become visible after a user clicks the “Buy” button (conditional rendering), and it should not be visible before that action is taken.
- On your `About` component, make use of the React Bootstrap **figures component** to display:
    - your store's logo (image),
    - a short description of your store,
    - two fictional images of your store, and
    - how to contact you.
- Your app must be styled as attractively as possible. You can use custom CSS or other React UI libraries to achieve this.

Be sure to place files for submission inside your task folder and click “Request review” on your dashboard.

Share your thoughts

HyperionDev strives to provide internationally excellent course content that helps you achieve your learning outcomes.

Do you think we’ve done a good job or do you think the content of this task, or this course as a whole, can be improved?

Share your thoughts anonymously using this **form**.