



TASK

React - Routing

Visit our website

Introduction

WELCOME TO THE REACT - BASIC ROUTING TASK!

HTML anchor tags (`<a>`) allow us to display different pages based on the URL. In HTML, anchor tags are commonly used to create hyperlinks that link to different web pages or resources. When you click on an anchor tag with an appropriate URL, the browser navigates to the linked page or resource.

React is typically used for building single-page applications (SPAs), where we have only one 'index.html' file. However, we may still want to build a multi-page-like application and navigate between different pages based on the URL. Instead of loading a new HTML page for each link, React uses routes to render different components based on the URL. These routes map URLs to specific components, and when a user clicks on a link or enters a specific URL, the React app dynamically updates the content without a full page refresh. React Router is used to create these routes that navigate and display specific components.



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 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.

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

Step 1: Navigate to React application

Open your command line interface (CLI) and navigate to your React application.

Step 2: Install react-router-dom

Type `npm install --save react-router-dom@latest` in the command line interface.

Step 3: Import BrowserRouter

Open `src/index.js` and import `{ BrowserRouter }` from `react-router-dom`.

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

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

The **BrowserRouter** is a component provided by React Router and is responsible for handling routing functionality in a React application. This feature helps with navigating and displaying content based on the URL. In a typical React application, the `index.js` component serves as the root component. Inside `index.js`, you would place the **BrowserRouter** as the parent and define the `App.js` component as its child. The routes will then be defined inside the `App` component as its children. Each route will correspond to a specific URL and determines which component to render. This allows us to create a multi-page application using components and display them based on the URL.

Step 4: Route tags

Open `src/App.js` and replace the default markup with some routes as seen below:

```
import { Routes, Route } from 'react-router-dom';
import Header from './components/Heading';
import Landing from './components/Landing';
import ShoppingCart from './components/ShoppingCart';
```

```

export default function App() {
  return (
    <div className="App">
      <Header />

      { /* // ADD OTHER COMPONENTS HERE */ }

      <Routes>
        <Route exact path="/" element={<Landing />} />
        <Route path="/cart" element={<ShoppingCart />} />
      </Routes>
    </div>
  );
}

```

Step 5: Route Rules

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:

- `<Route path="/cart" element={<ShoppingCart />} />`; will display the **ShoppingCart** component on the screen when the URL in your browser's address bar contains the string `"/cart"`. e.g. `"http://localhost:3000/cart"`
- `<Route exact path="/" element={<Landing />} />`; will display the **Landing** 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:3000/"` instead. This would be your root URL.
 - The code `'exact'` specifies that the URL must be exactly the one specified, not just contain that value.
 - If `'exact'` were to be removed from the code in the example above, the URL `"cart"` would also display the **Landing** component because that URL also contains the string `"/"`.

Note: Components that you want to display on all pages regardless of the URL can be added without a **Route** tag. For example, the **Header** component will be displayed on all pages, regardless of the URL.



Extra resource

For more information about using React Router, read more in the [documentation](#).

After creating your components, you will import them into App.js. Here, some of your components will be called inside the **Route** elements to only be displayed on a specific URL. Note that it is recommended to place your components inside a components folder. e.g 'src/Components/Landing.js'

The following are examples of three individual components files. Create these files and place them inside your components folder.

```
// Landing.js component
import React from 'react'
import { Link } from "react-router-dom";

export default function Landing() {
  return (
    <div>
      <h2>Welcome to the homepage!</h2>
      <nav>
        <Link to="/cart">Cart</Link>
      </nav>
    </div>
  );
}
```

```
// ShoppingCart.js component
import React from 'react'
import { Link } from "react-router-dom";

export default function ShoppingCart() {
  return (
    <div>
      <h2>Shopping Cart</h2>
      <nav>
```

```

    <Link to="/">Home</Link>
  </nav>
</div>
);
}

```

```

// Header.js component
import React from 'react'

export default function Header() {
  return (
    <h1>This is the Header</h1>
  );
}

```

Step 6: Button Link

The **Link** component provides declarative, accessible navigation around your application. E.g. `<Link to="/">Home</Link>` will take the user to the Landing component when Home is clicked on. Note, that the Link element 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.

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, a global state manager like Redux, or local storage.

React router hooks

Router hooks can be used to pass the component's state value and store it in a separate component defined outside the routes. This way, the state remains intact as you navigate between different parts of your application. Storing state outside your routes can provide valuable information to other components in your

application. By keeping the state at a higher level and accessible from multiple components, you can easily share and utilise important data across various pages of 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, and 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 to other pages with **useNavigation()**:

```
1  import React from "react";
2  import { useNavigate } from "react-router-dom";
3
4  export default function Home() {
5    const nav = useNavigate();
6
7    return (
8      <div className="tab-content">
9        <button
10          onClick={() => nav(`/products`,
11            { state: { product: "iPhone 12" } })}
12        >Buy</button>
13      </div>
14    );
15 }
```

- 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 12"` (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

```
1 import React from "react";
2 import { useLocation } from "react-router-dom";
3
4 export default function Home() {
5   const location = useLocation();
6   const data = location.state;
7   console.log('product', data.product); // Outputs: 'iPhone 12'
8
9   return (
10     <h3>Product page</h3>
11   );
12 }
```

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

- Line 2 imports the `useLocation` hook from `"react-router-dom"`.
- Line 5 initialises the `location` constant by calling the `useLocation` hook.
- Line 6 initialises the `data` constant with the state object.
- Line 7 displays the product value in the log.

State management alternatives

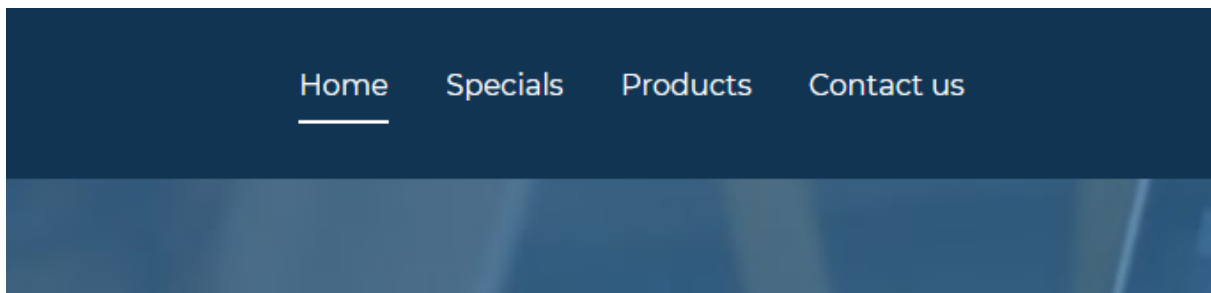
An alternative to React Router hooks would be to use local storage. If the data is not sensitive and doesn't need to be shared with other components in real-time, you can store the data in the browser's **localStorage**. This method allows you to store data on the client side, which will persist when navigating to a new page.

Another option is to use **Redux** and **React global state management** hooks. These allow you to create a global state that can be accessed from any component in your application, regardless of the route.


REACT ROUTER LIBRARY

The React Router library comes with a number of useful prebuilt hooks. For example, if you want to render content like an image or text based on the URL, 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 do 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, e.g. www.hyperiondev.com.



Take note:

To learn more about Router Hooks and build a more robust and intuitive application, check out the [source documentation](#).

Compulsory 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:

- 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 a `<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.

Please submit the code for your multi-page application.

Compulsory Task 2

Follow these steps:

- In your Products component, display at least 10 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 3 colour options. Use a Bootstrap [dropdown button](#) for this.
- When a colour option is selected inside the dropdown button, the text that says “Dropdown button” should change to that selected colour.
- Create a component called **TotalPrice.js** 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, 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.



Rate us

Share your thoughts

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

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

[Click here](#) to share your thoughts anonymously.

