前端網站開發人員證書課程
(二) 進階網絡程式設計--專業React.js應用

6. React Router

Presented by Krystal Institute

Lesson Outline

- Introduce React router, which enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL
- Discuss the different types of routers available and what router components and types are, and the important router terminologies used in react-router.

6.1 React as Single Page Application (SPA)

6.1 React as Single Page Application (SPA)

- Before diving into the routing, you need to understand how pages are rendered in React app.
- In non-single page applications, when you click on a link in the browser, a request is sent to the server before the HTML page gets rendered.
- In React, the page contents are created from our components. So what React Router does is intercept the request being sent to the server and then inject the contents dynamically from the components we have created. This is the general idea behind SPAs which allows content to be rendered faster without the page being refreshed.

6.1 React as Single Page Application (SPA)

- When you create a new project, you'll always see an *index.html* file in the public folder. All the code you write in your *App* component which acts as the root component gets rendered to this HTML file. This means that there is only one HTML file where your code will be rendered to.
- What happens when you have a different component you would prefer to render as a different page? Do you create a new HTML file? The answer is no. React Router – as the name implies – helps you route to/navigate to and render your new component in the *index.html* file.
- So as a single-page application, when you navigate to a new component using React Router, the *index.html* will be rewritten with the component's logic.

6.2 React Router

6.2 React Router

- React Router is a fully-featured client and server-side routing library for React, a JavaScript library for building user interfaces.
- React Router runs anywhere React runs.
- React Router is compatible with React >= 16.8.
- React Router isn't just about matching a url to a function or component: it's about building a full user interface that maps to the URL, so it might have more concepts in it than you're used to.
- Three main jobs of React Router:
    - Subscribing and Manipulating the *history stack*.
    - Matching the *URL* to your *routes*.
    - Rendering a nested UI from the *route matches*.

6.2 React Router

History stack:

- As the user navigates, the browser keeps track of each *location* in a stack.
- If you click and hold the back button in a browser you can see the browser's history stack right there.

Location:

- This is a React Router specific object that is based on the built-in browser's window.location object.
- It represents "where the user is at". It's mostly an object representation of the *URL* but has a bit more to it than that.

6.2 React Router

URL:

- The URL is in the address bar.
- A lot of people use the term "*URL*" and "*route*" interchangeably, but this is not a route in React Router, it's just a *URL*.

Route:

- An object or Route Element typically with a shape of { path, element } or <Route path element>.
- The path is a path pattern.
- When the path pattern matches the current URL, the element will be rendered.

6.2 React Router

Matches:

- An object that holds information when a route matches the URL, like the *url params* and pathname that matched.

URL Params:

- The parsed values from the URL matched a dynamic segment.
- A dynamic segment is a segment of a path pattern that is dynamic, meaning it can match any values in the segment.
- Example: The pattern /users/:userId will match URLs like /users/123

6.3 Router

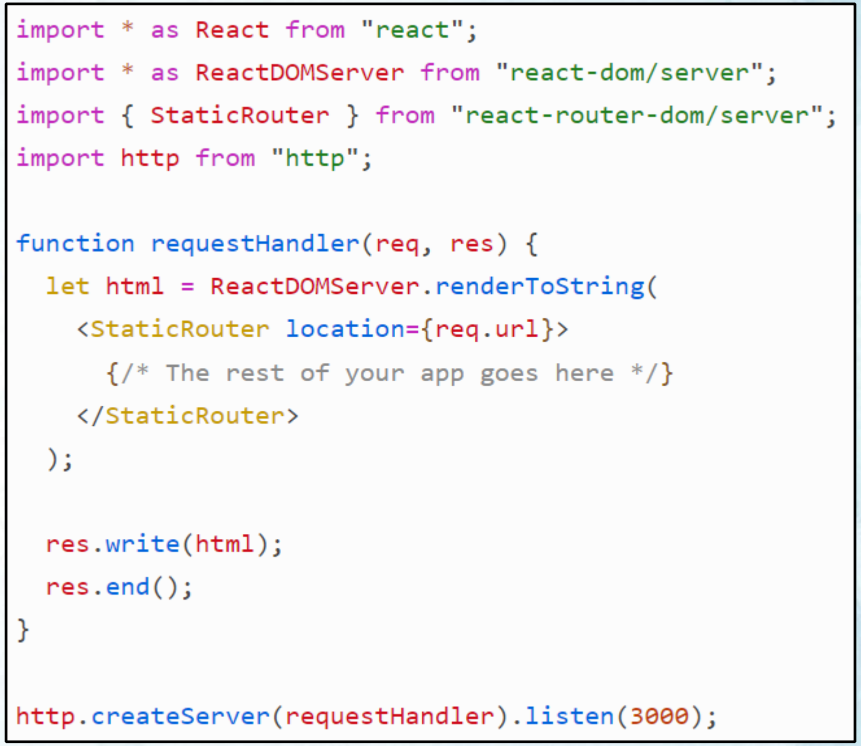
6.3 Router

- *<Router>* is the low-level interface that is shared by all router components (like *<BrowserRouter>* and *<StaticRouter>*).
- In terms of React, *<Router>* is a context provider that supplies routing information to the rest of the app.
- You probably never need to render a *<Router>* manually. Instead, you should use one of the higher-level routers depending on your environment. You only ever need one router in a given app.
- The *<Router basename>* prop may be used to make all routes and links in your app relative to a "base" portion of the URL pathname that they all share.
- This is useful when rendering only a portion of a larger app with React Router or when your app has multiple entry points. Basenames are not case-sensitive.

6.3 Router

Static Router

- *<StaticRouter>* is used to render a React Router web app in node. Provide the current location via the *location* prop.
- *<StaticRouter location>* defaults to *"/"*



```
import * as React from "react";
import * as ReactDOMServer from "react-dom/server";
import { StaticRouter } from "react-router-dom/server";
import http from "http";

function requestHandler(req, res) {
  let html = ReactDOMServer.renderToString(
    <StaticRouter location={req.url}>
      {/* The rest of your app goes here */}
    </StaticRouter>
  );

  res.write(html);
  res.end();
}

http.createServer(requestHandler).listen(3000);
```

6.3 Router

Browser Router

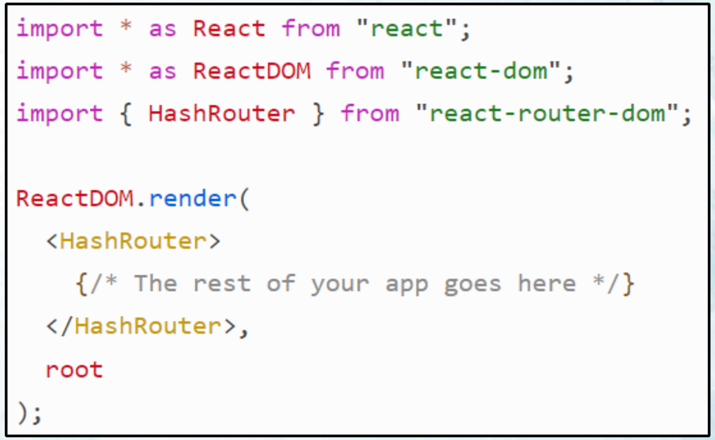
- *<BrowserRouter>* is the recommended interface for running React Router in a web browser.
- A *<BrowserRouter>* stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.
- *<BrowserRouter window>* defaults to using the current document's default View, but it may also be used to track changes to another window's URL, in an <iframe>

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

ReactDOM.render(
  <BrowserRouter>
    {/* The rest of your app goes here */}
  </BrowserRouter>,
  root
);
```

6.3 Router

Hash Router

- *<HashRouter>* is for use in web browsers when the URL should not (or cannot) be sent to the server for some reason.
- This may happen in some shared hosting scenarios where you do not have full control over the server.
- In these situations, *<HashRouter>* makes it possible to store the current location in the hash portion of the current URL, so it is never sent to the server.
- *<HashRouter window>* defaults to using the current document's defaultView, but it may also be used to track changes to another window's URL, in an <iframe>

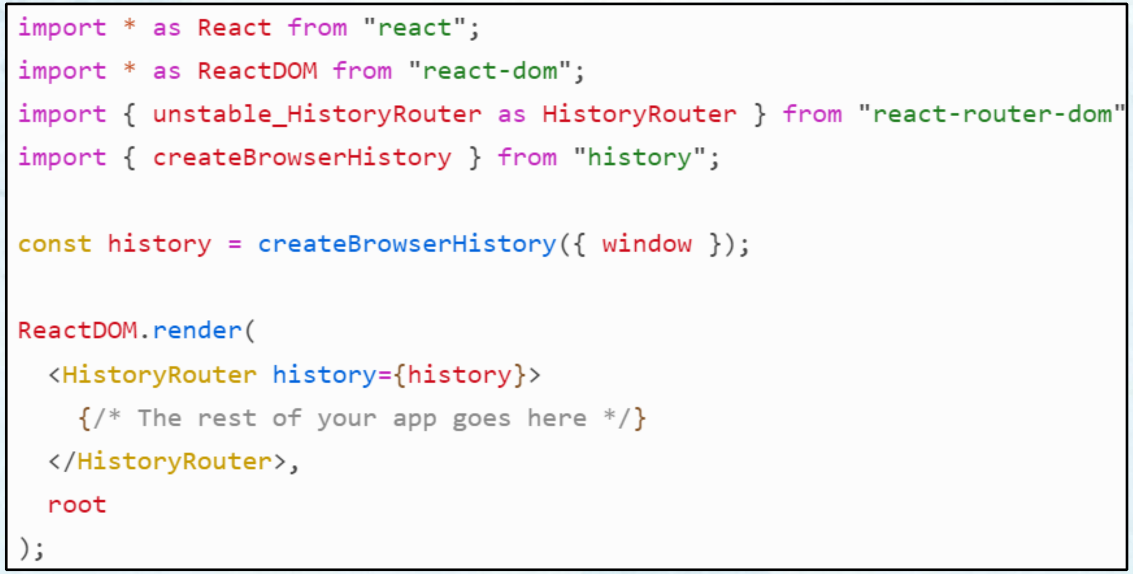


```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { HashRouter } from "react-router-dom";

ReactDOM.render(
  <HashRouter>
    {/* The rest of your app goes here */}
  </HashRouter>,
  root
);
```

6.3 Router

Unstable History Router

- *<unstable_HistoryRouter>* takes an instance of the history library as a prop.
- This allows you to use that instance in non-React contexts or as a global variable.



```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { unstable_HistoryRouter as HistoryRouter } from "react-router-dom"
import { createBrowserHistory } from "history";

const history = createBrowserHistory({ window });

ReactDOM.render(
  <HistoryRouter history={history}>
    {/* The rest of your app goes here */}
  </HistoryRouter>,
  root
);
```

6.3 Router

Memory Router

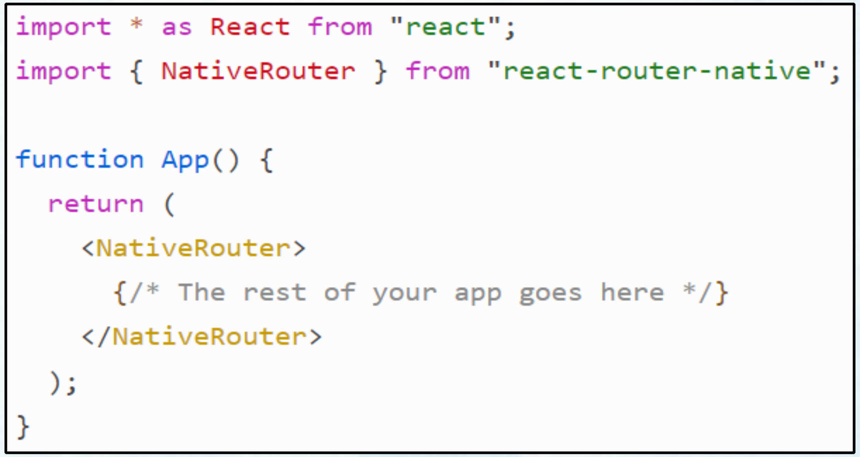
- A *<MemoryRouter>* stores its locations internally in an array. Unlike *<BrowserHistory>* and *<HashHistory>*, it isn't tied to an external source, like the history stack in a browser.
- This makes it ideal for scenarios where you need complete control over the history stack, like testing.
  - *<MemoryRouter initialEntries>* defaults to *["/"]* (a single entry at the root / URL)
  - *<MemoryRouter initialIndex>* defaults to the last index of *initialEntries*.

```
import * as React from "react";
import { create } from "react-test-renderer";
import {
  MemoryRouter,
  Routes,
  Route,
} from "react-router-dom";

describe("My app", () => {
  it("renders correctly", () => {
    let renderer = create(
      <MemoryRouter initialEntries={["/users/mjackson"]}>
        <Routes>
          <Route path="users" element={<Users />}>
            <Route path=":id" element={<UserProfile />} />
          </Route>
        </Routes>
      </MemoryRouter>
    );

    expect(renderer.toJSON()).toMatchSnapshot();
  });
});
```

6.3 Router

Native Router

- *<NativeRouter>* is the recommended interface for running React Router in a React Native app.
  - *<NativeRouter initialEntries>* defaults to *["/"]* (a single entry at the root / URL)
  - *<NativeRouter initialIndex>* defaults to the last index of *initialEntries*.



```
import * as React from "react";
import { NativeRouter } from "react-router-native";

function App() {
  return (
    <NativeRouter>
      {/* The rest of your app goes here */}
    </NativeRouter>
  );
}
```

6.4 Components

6.4 Components

- Any component passed as a child to *<Router>* is called a “*Route Component*”.
- There are three types of props for Route Components.
- Matching Props - You provide these props where the *<Router>* is rendered.
- They are used by the Router to match the component against the location to see if the component should be rendered.

6.4 Components

Link

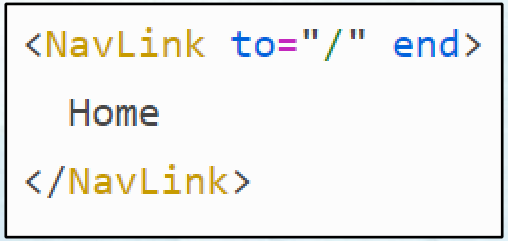
- A *<Link>* is an element that lets the user navigate to another page by clicking or tapping on it.
- In react-router-dom, a *<Link>* renders an accessible *<a>* element with a real href that points to the resource it's linking to.
- This means that things like right-clicking a *<Link>* work as you'd expect.
- You can use *<Link reloadDocument>* to skip client side routing and let the browser handle the transition normally (as if it were an *<a href>*).

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

function UsersIndexPage({ users }) {
  return (
    <div>
      <h1>Users</h1>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            <Link to={user.id}>{user.name}</Link>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

6.4 Components

NavLink

- A *<NavLink>* is a special kind of *<Link>* that knows whether or not it is "*active*".
- This is useful when building a navigation menu such as a breadcrumb or a set of tabs where you'd like to show which of them is currently selected.
- It also provides useful context for assistive technology like screen readers.
- By default, an active class is added to a *<NavLink>* component when it is active.
- Instead, you can pass a function to either style or *className* that will allow you to customize the inline styling or the class string based on the component's active state.
- You can also pass a function as children to customize the content of the *<NavLink>* component based on their active state, especially useful to change styles on internal elements.



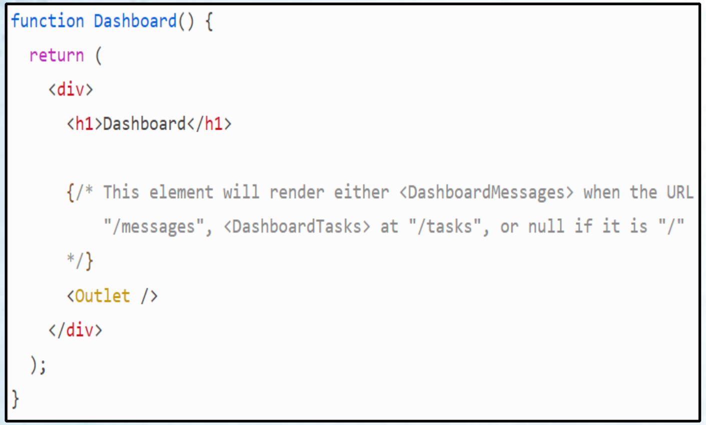
```
<NavLink to="/" end>
  Home
</NavLink>
```

6.4 Components

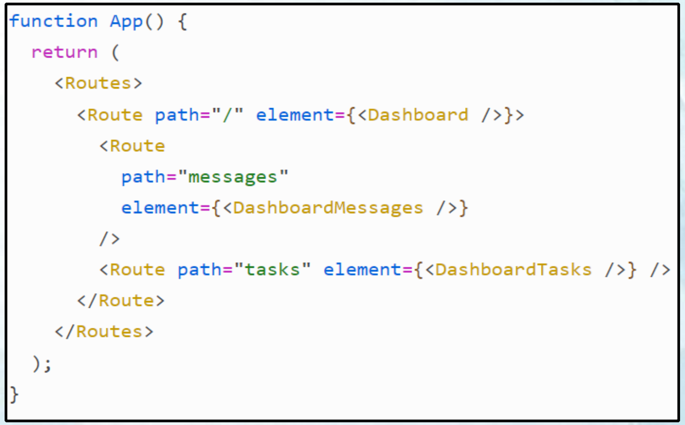
Navigate & Outlet

- A *<Navigate>* element changes the current location when it is rendered.
- It's a component wrapper around *useNavigate*, and accepts all the same arguments as props.

Outlet: An *<Outlet>* should be used in parent route elements to render their child route elements.



```
function Dashboard() {
  return (
    <div>
      <h1>Dashboard</h1>

      {/* This element will render either <DashboardMessages> when the URL
          "/messages", <DashboardTasks> at "/tasks", or null if it is "/"
      */}
      <Outlet />
    </div>
  );
}
```

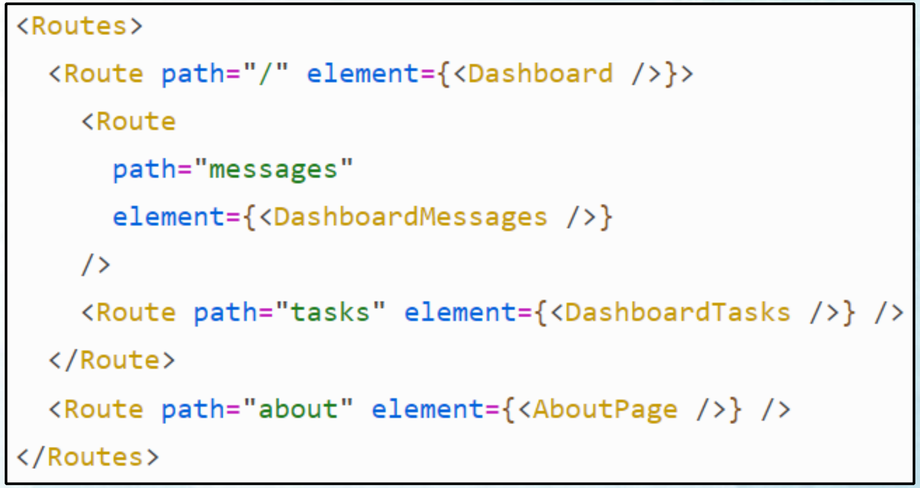


```
function App() {
  return (
    <Routes>
      <Route path="/" element={<Dashboard />}>
        <Route
          path="messages"
          element={<DashboardMessages />}
        />
        <Route path="tasks" element={<DashboardTasks />} />
      </Route>
    </Routes>
  );
}
```

6.4 Components

Routes and Route

- *<Routes>* and *<Route>* are the primary ways to render something in React Router based on the current location.
- You can think about a *<Route>* kind of like an if statement; if its path matches the current URL, it renders its *element*.
- The *<Route caseSensitive>* prop determines if the matching should be done in a case-sensitive manner (defaults to false).
- Whenever the location changes, *<Routes>* looks through all its children *<Route>* elements to find the best match and renders that branch of the UI.
- *<Route>* elements may be nested to indicate nested UI, which also corresponds to nested URL paths. Parent routes render their child routes by rendering an *<Outlet>*.



```
<Routes>
  <Route path="/" element={<Dashboard />}>
    <Route
      path="messages"
      element={<DashboardMessages />}
    />
    <Route path="tasks" element={<DashboardTasks />} />
  </Route>
  <Route path="about" element={<AboutPage />} />
</Routes>
```

6.5 Building Bookkeeping App

6.5 Building Bookkeeping App

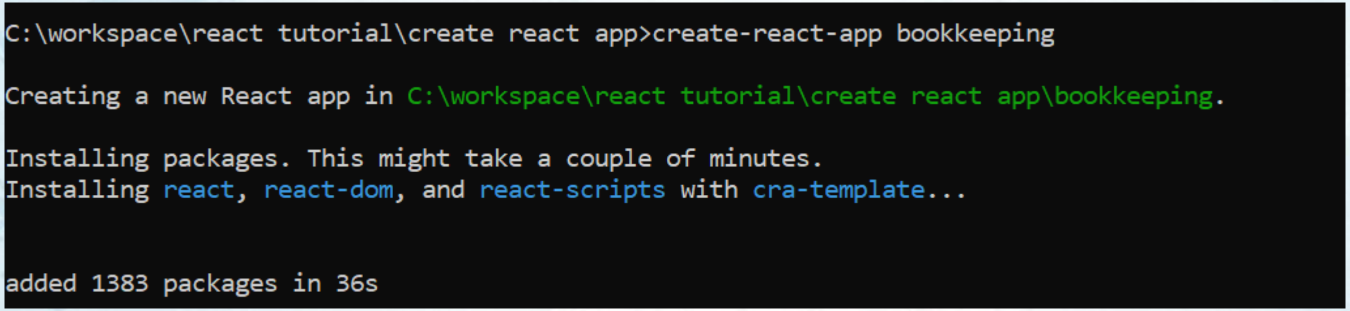
While building a little bookkeeping app we will learn the following:

- Configuring routes.
- Navigating with a link.
- Creating links with active styling.
- Using nested routes for layout.
- Navigating programmatically.

6.5 Building Bookkeeping App

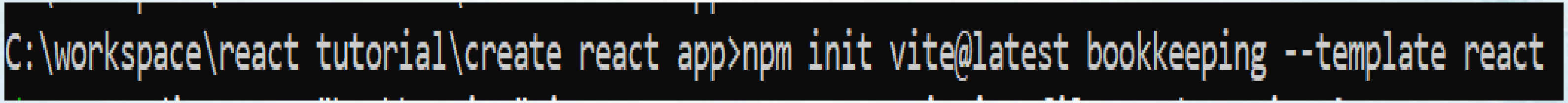
Installation

- Create a react app before installing the react router.



```
C:\workspace\react tutorial\create react app>create-react-app bookkeeping

Creating a new React app in C:\workspace\react tutorial\create react app\bookkeeping.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1383 packages in 36s
```

- Install the router.

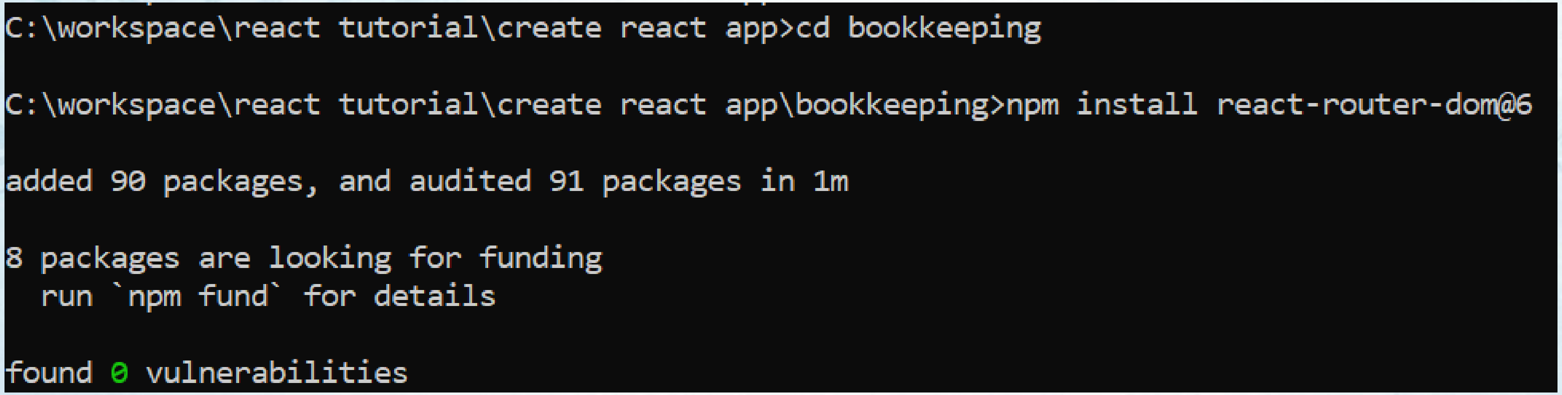


```
C:\workspace\react tutorial\create react app>npm init vite@latest bookkeeping --template react
```

6.5 Building Bookkeeping App

Installation

- Get into the app folder and install the router dependencies.

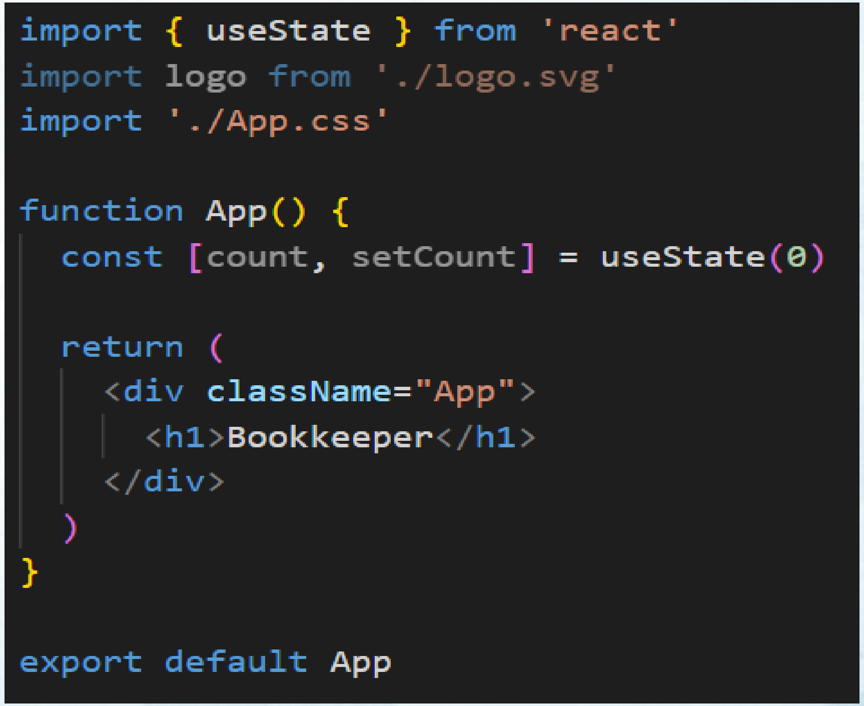


```
C:\workspace\react tutorial\create react app>cd bookkeeping

C:\workspace\react tutorial\create react app\bookkeeping>npm install react-router-dom@6

added 90 packages, and audited 91 packages in 1m

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

6.5 Building Bookkeeping App

Installation

- Open the app in the Visual Studio Code and edit your App.js

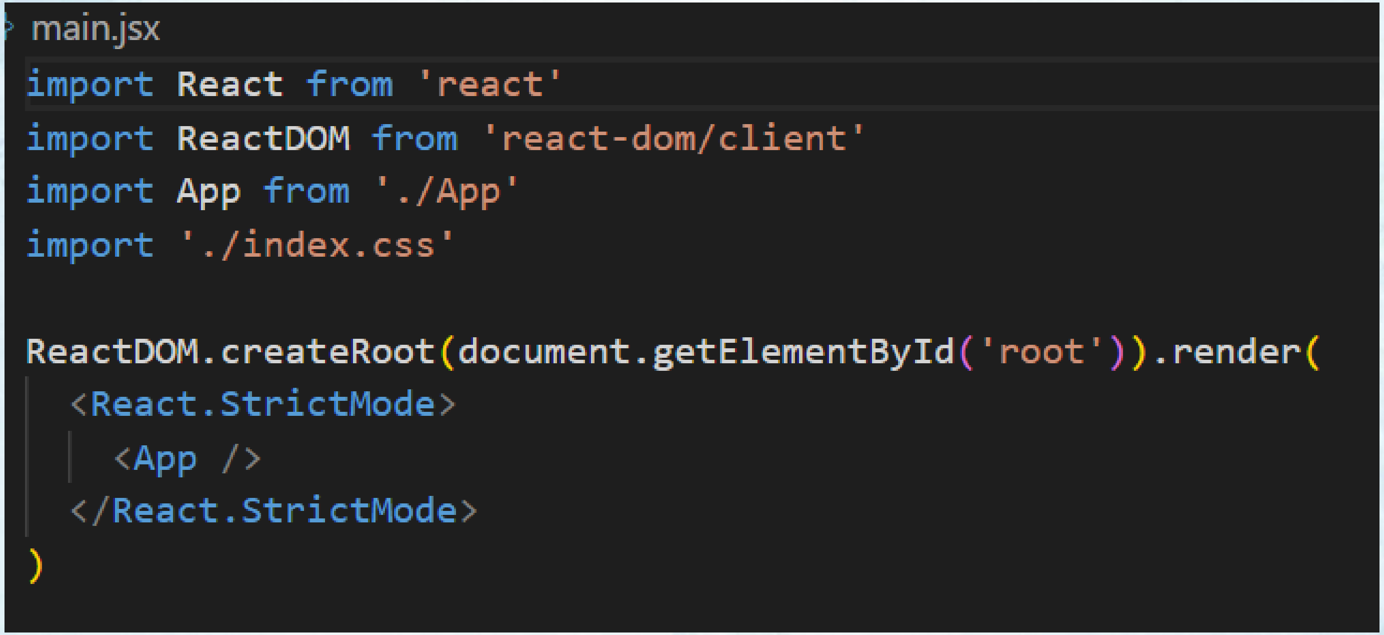


```
import { useState } from 'react'
import logo from './logo.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div className="App">
      <h1>Bookkeeper</h1>
    </div>
  )
}

export default App
```

6.5 Building Bookkeeping App

Installation

- Make sure the App component is rendered in main.jsx



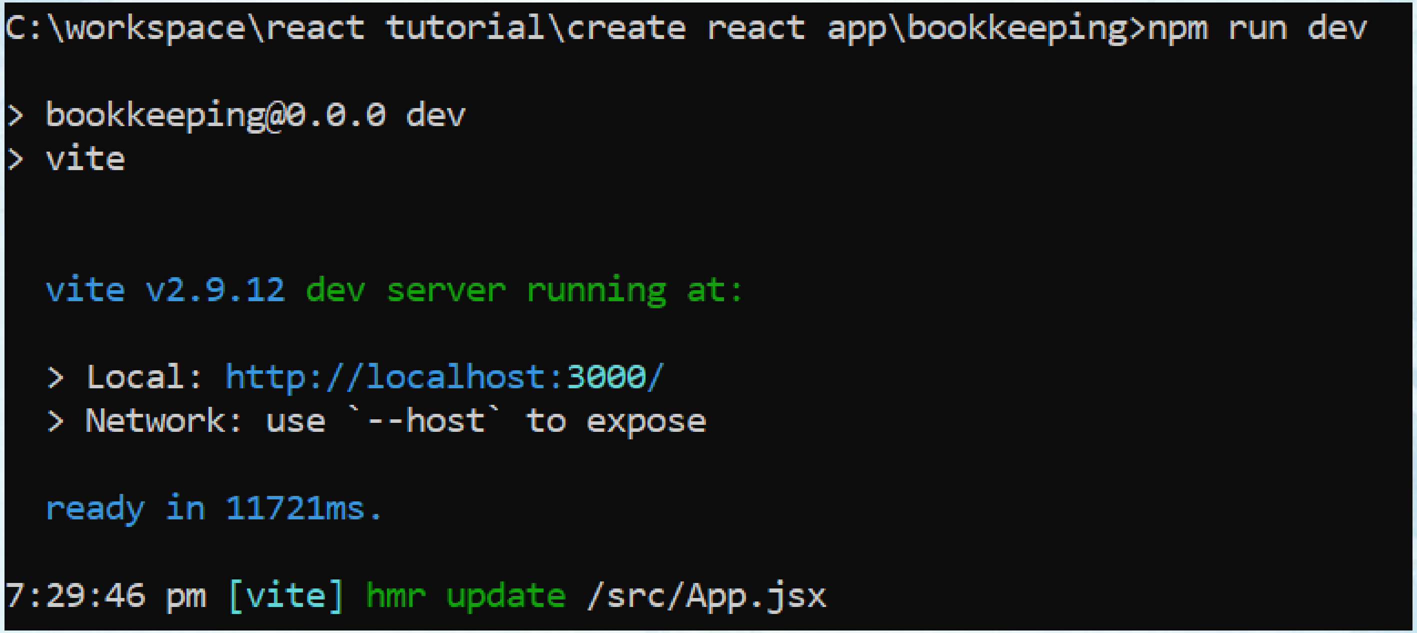
main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

6.5 Building Bookkeeping App

Installation

- Start your app.

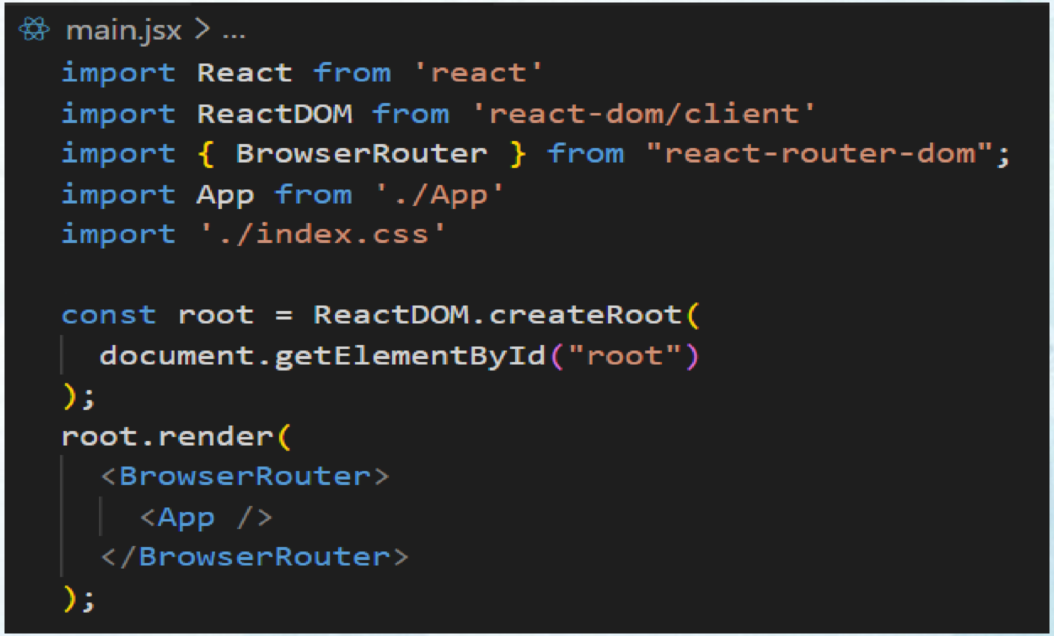


```
C:\workspace\react tutorial\create react app\bookkeeping>npm run dev

> bookkeeping@0.0.0 dev
> vite


  vite v2.9.12 dev server running at:

  > Local: http://localhost:3000/
  > Network: use `--host` to expose

  ready in 11721ms.

7:29:46 pm [vite] hmr update /src/App.jsx
```

6.5 Building Bookkeeping App

Installation

- First things first, we want to connect your app to the browser's URL: import BrowserRouter and render it around your whole app.



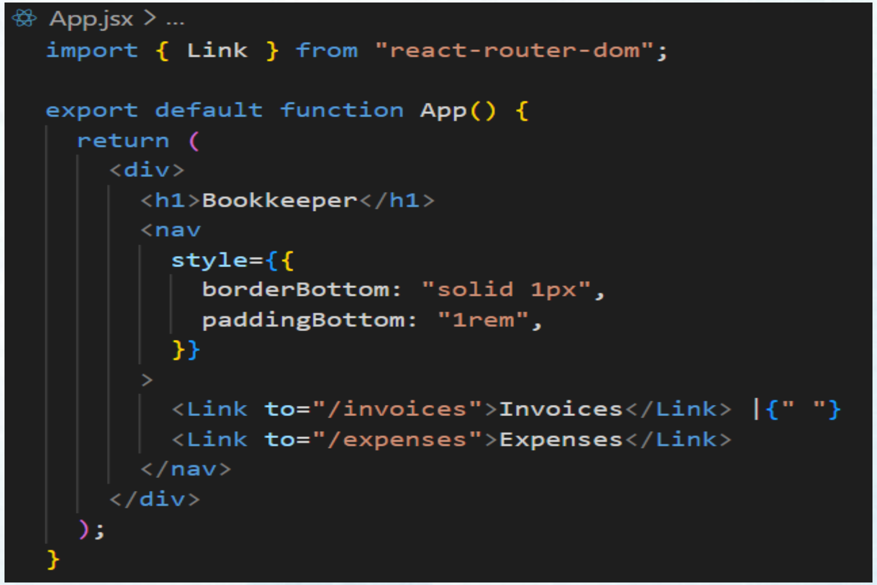
main.jsx > ...

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

const root = ReactDOM.createRoot(
  document.getElementById("root")
);
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

6.5 Building Bookkeeping App

Installation

- Nothing changes in your app, but now we're ready to start messing with the URL.
- Open up src/App.js, import Link and add some global navigation.



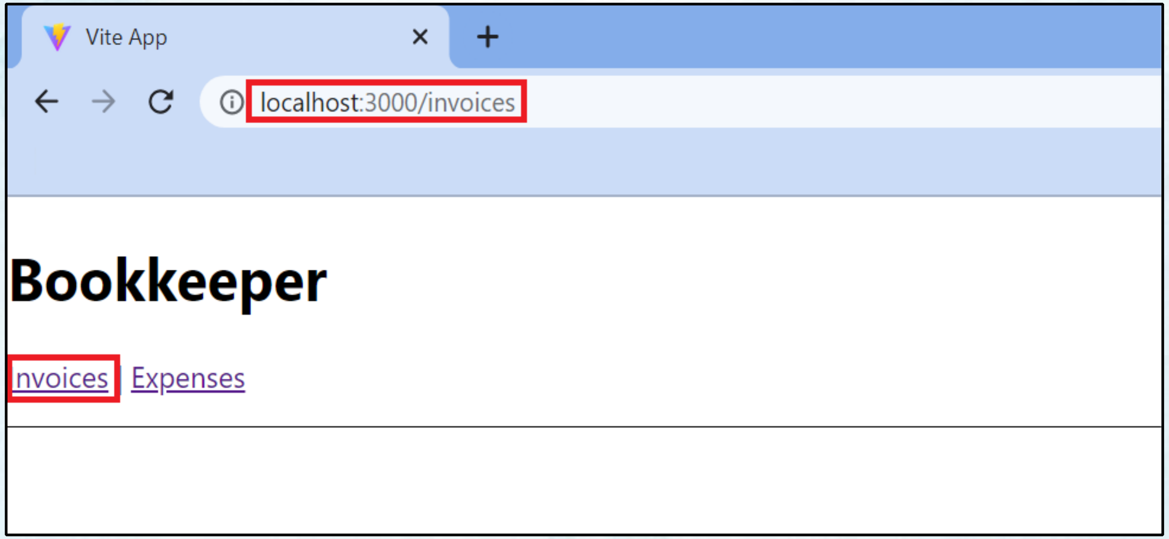
App.jsx > ...

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

export default function App() {
  return (
    <div>
      <h1>Bookkeeper</h1>
      <nav
        style={{
          borderBottom: "solid 1px",
          paddingBottom: "1rem",
        }}
      >
        <Link to="/invoices">Invoices</Link> |{" "}
        <Link to="/expenses">Expenses</Link>
      </nav>
    </div>
  );
}
```

6.5 Building Bookkeeping App

Installation

- View the output, Go ahead and click the links.
- When Invoices is clicked.



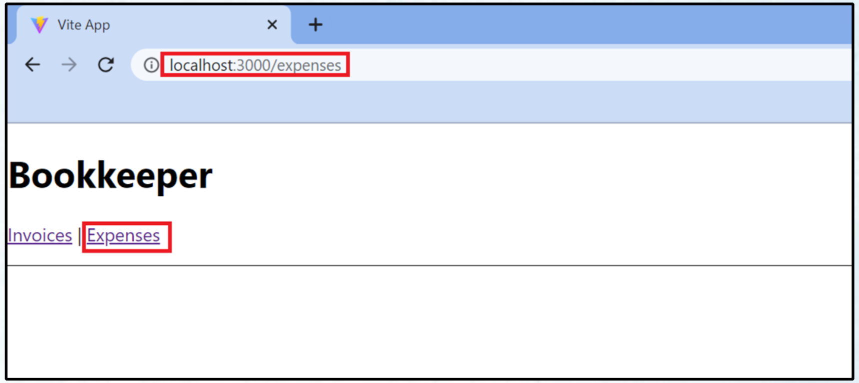
Bookkeeper

nvoices | Expenses

6.5 Building Bookkeeping App

Installation

- When Expenses is clicked.



Bookkeeper

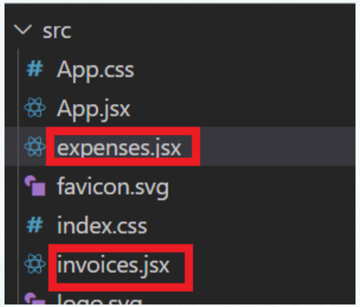
Invoices | Expenses

- You'll need to click the "Open in New Window" button in the inline-browser's toolbar). React Router is now controlling the URL!
- We don't have any routes that render when the URL changes yet, but Link is changing the URL without causing a full page reload.

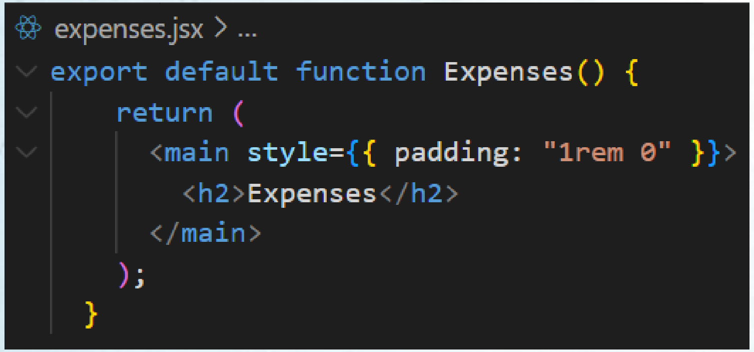
6.6 Routes

6.6 Routes

- Add a couple new files: invoices.jsx and expenses.jsx and fill them with some basic code.

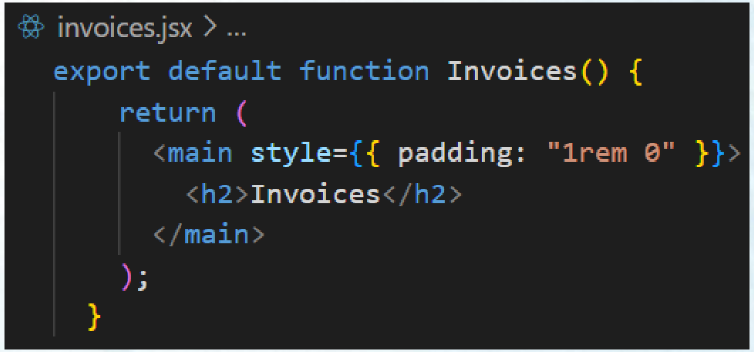


```
src
  App.css
  App.jsx
  expenses.jsx
  favicon.svg
  index.css
  invoices.jsx
  logo.svg
```



expenses.jsx > ...

```jsx
export default function Expenses() {
  return (
    <main style={{ padding: "1rem 0" }}>
      <h2>Expenses</h2>
    </main>
  );
}
```

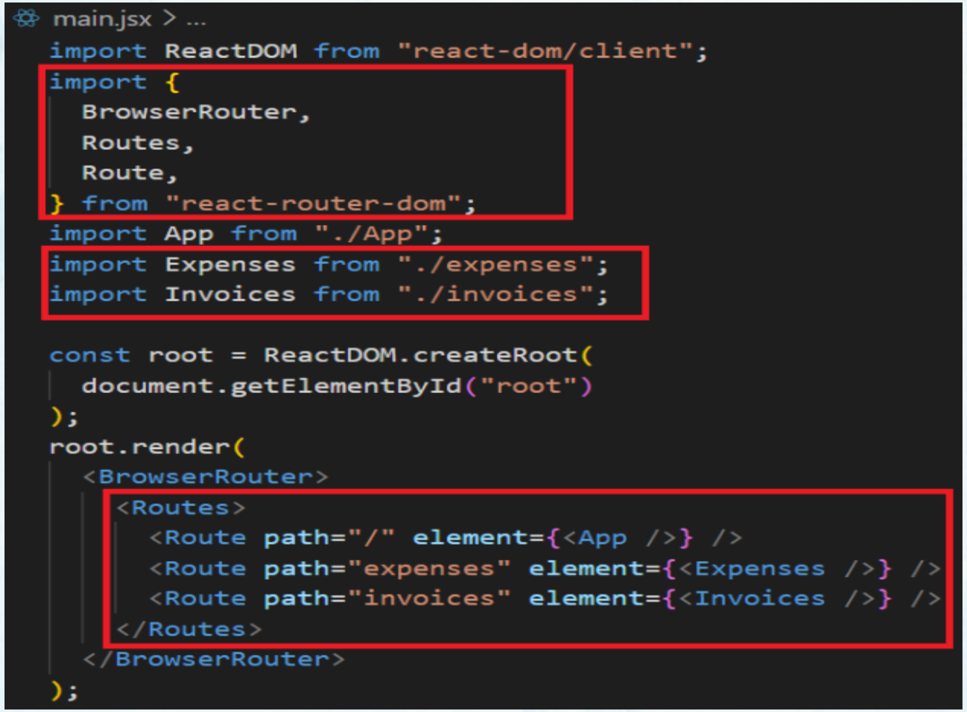


invoices.jsx > ...

```jsx
export default function Invoices() {
  return (
    <main style={{ padding: "1rem 0" }}>
      <h2>Invoices</h2>
    </main>
  );
}
```

6.6 Routes

- Let's render our app at different URLs by creating our first "Route Config" inside of main.jsx

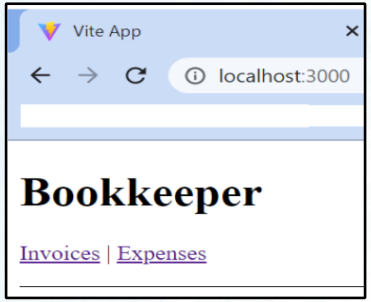


main.jsx > ...

```
import ReactDOM from "react-dom/client";
import {
  BrowserRouter,
  Routes,
  Route,
} from "react-router-dom";
import App from "./App";
import Expenses from "./expenses";
import Invoices from "./invoices";

const root = ReactDOM.createRoot(
  document.getElementById("root")
);
root.render(
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<App />} />
      <Route path="expenses" element={<Expenses />} />
      <Route path="invoices" element={<Invoices />} />
    </Routes>
  </BrowserRouter>
);
```

6.6 Routes

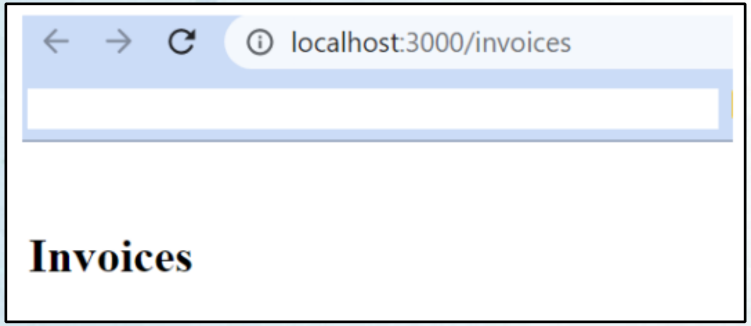
- Notice at "/" it renders <App>. At "/invoices" it renders <Invoices>.
- View the output, open the links in the new tab and verify the code.


Vite App

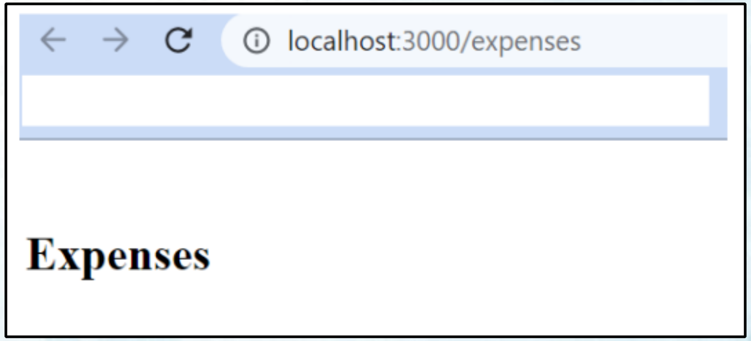
localhost:3000

Bookkeeper

Invoices | Expenses


localhost:3000/invoices

Invoices

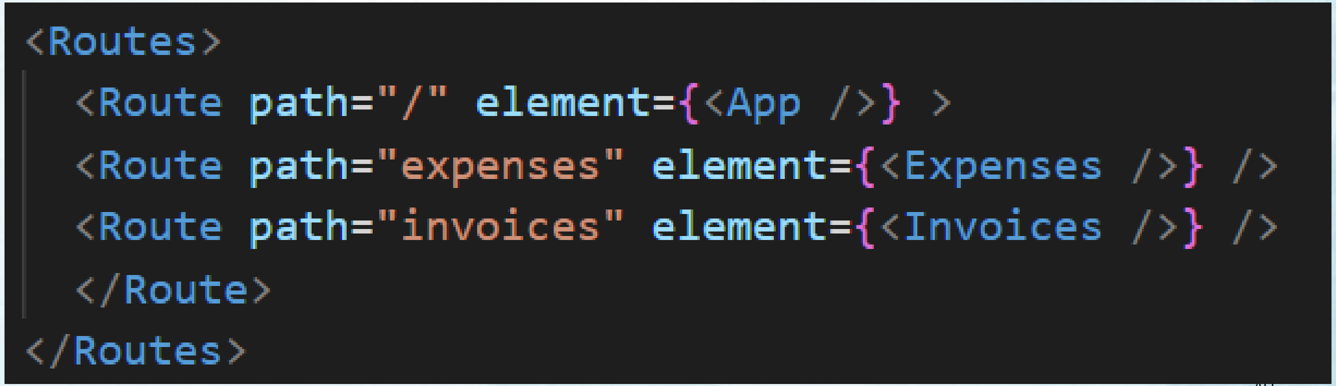

localhost:3000/expenses

Expenses

6.6 Routes

Nested Routes

- Let's get some automatic, persistent layout handling by doing just two things:
    - Nest the routes inside of the App route
    - Render an Outlet
- First let's nest the routes. Right now the expenses and invoices routes are siblings to the app, we want to make them children of the app route. Make changes in the main.jsx file:

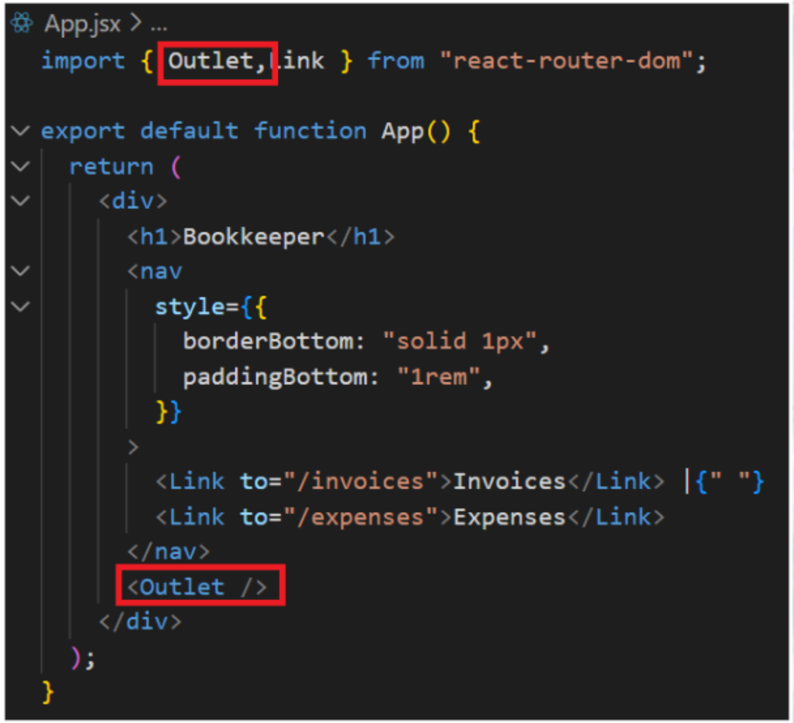


```
<Routes>
  <Route path="/" element={<App />} >
  <Route path="expenses" element={<Expenses />} />
  <Route path="invoices" element={<Invoices />} />
  </Route>
</Routes>
```

6.6 Routes

Nested Routes

- Render an Outlet in the App.jsx "parent" route.
- When routes have children it does two things:
    - It nests the URLs ("/" + "expenses" and "/" + "invoices")
    - It will nest the UI components for shared layout when the child route matches.

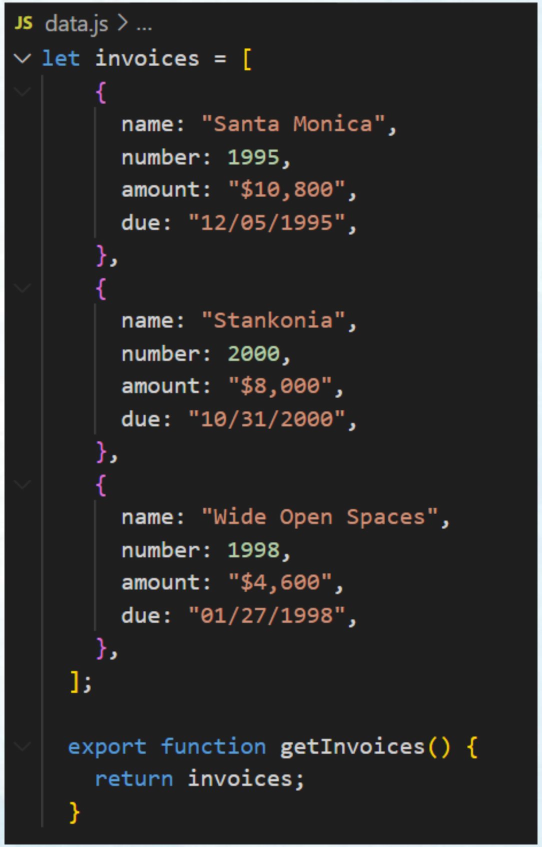

App.jsx > ...

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

export default function App() {
  return (
    <div>
      <h1>Bookkeeper</h1>
      <nav
        style={{
          borderBottom: "solid 1px",
          paddingBottom: "1rem",
        }}
      >
        <Link to="/invoices">Invoices</Link> |{" "}
        <Link to="/expenses">Expenses</Link>
      </nav>
      <Outlet />
    </div>
  );
}
```

6.6 Routes

Listing the invoices

- Make a new file data.js and hardcode some fake stuff so that we can focus on routing. Normally the data will be fetched from the server.



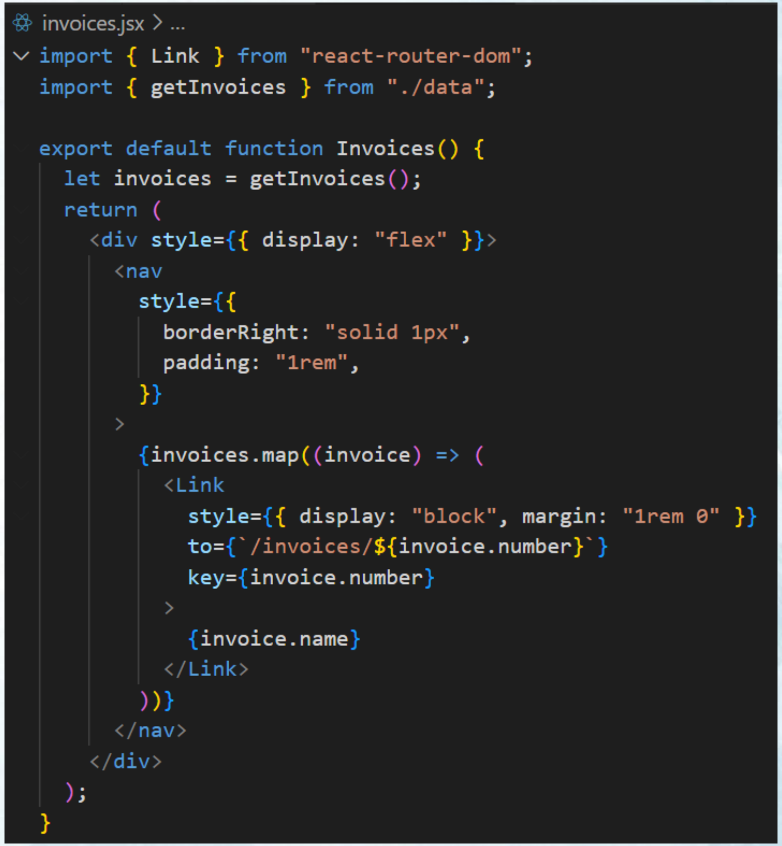
data.js > ...

```
let invoices = [
    {
      name: "Santa Monica",
      number: 1995,
      amount: "$10,800",
      due: "12/05/1995",
    },
    {
      name: "Stankonia",
      number: 2000,
      amount: "$8,000",
      due: "10/31/2000",
    },
    {
      name: "Wide Open Spaces",
      number: 1998,
      amount: "$4,600",
      due: "01/27/1998",
    },
  ];

  export function getInvoices() {
    return invoices;
  }
```

6.6 Routes

Listing the invoices

- Use the data in invoices route.



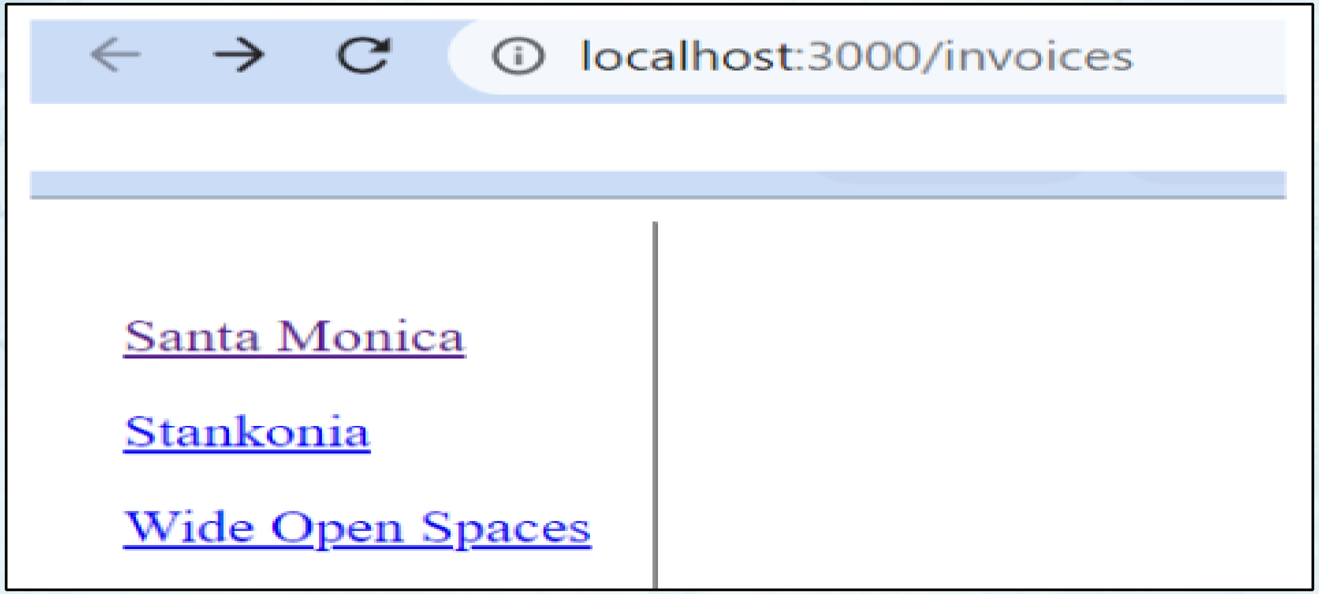
invoices.jsx > ...

```
import { Link } from "react-router-dom";
import { getInvoices } from "./data";

export default function Invoices() {
  let invoices = getInvoices();
  return (
    <div style={{ display: "flex" }}>
      <nav
        style={{
          borderRight: "solid 1px",
          padding: "1rem",
        }}
      >
        {invoices.map((invoice) => (
          <Link
            style={{ display: "block", margin: "1rem 0" }}
            to={`/invoices/${invoice.number}`}
            key={invoice.number}
          >
            {invoice.name}
          </Link>
        ))}
      </nav>
    </div>
  );
}
```

6.6 Routes

Listing the invoices

- View the output and click an invoice link and see what happens.



Santa Monica

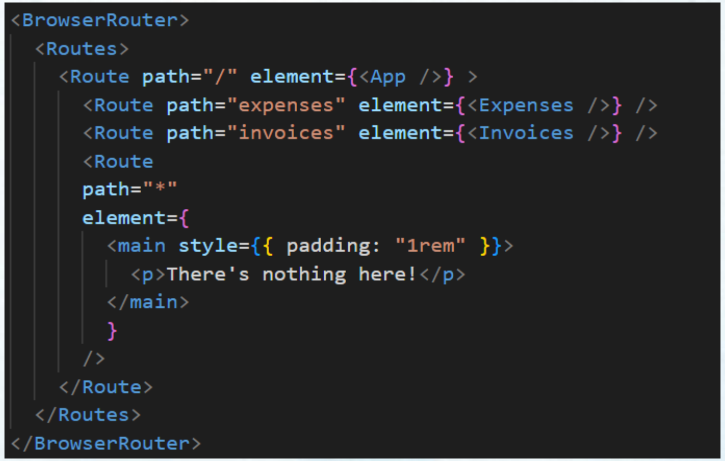
Stankonia

Wide Open Spaces

6.6 Routes

Adding a “No Match” Route

- If you click those links in the invoices page it goes blank! That's because none of the routes we've defined match a URL like the ones we're linking to: "/invoices/123".
- It's good practice to always handle this "no match" case. Go back to your config and add.

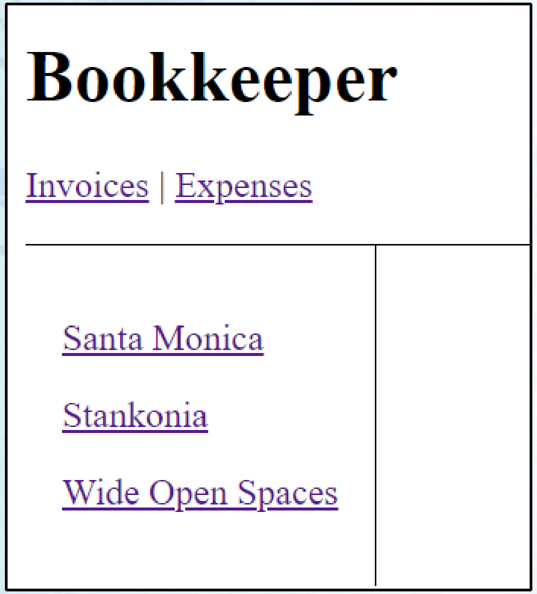


```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<App />} >
      <Route path="expenses" element={<Expenses />} />
      <Route path="invoices" element={<Invoices />} />
      <Route
      path="*"
      element={
        <main style={{ padding: "1rem" }}>
          <p>There's nothing here!</p>
        </main>
        }
      />
    </Route>
  </Routes>
</BrowserRouter>
```

6.6 Routes

Adding a “No Match” Route

- View and verify the output by clicking the links.



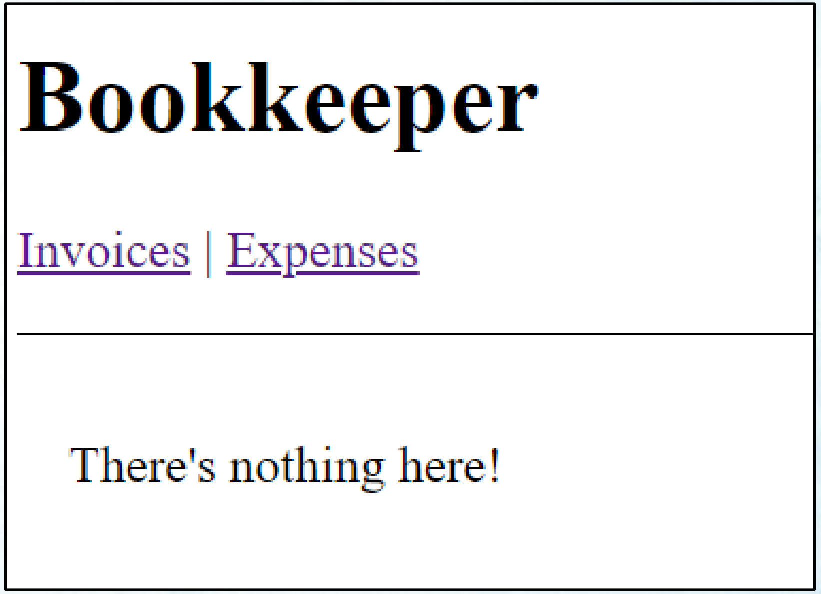
Bookkeeper

Invoices | Expenses

Santa Monica

Stankonia

Wide Open Spaces



Bookkeeper

Invoices | Expenses

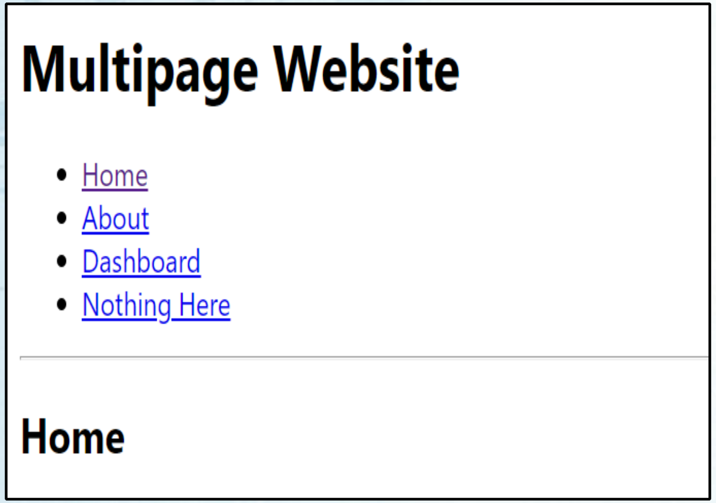
There's nothing here!

6.7 Assignment

6.7 Assignment

<u>Outline:</u>

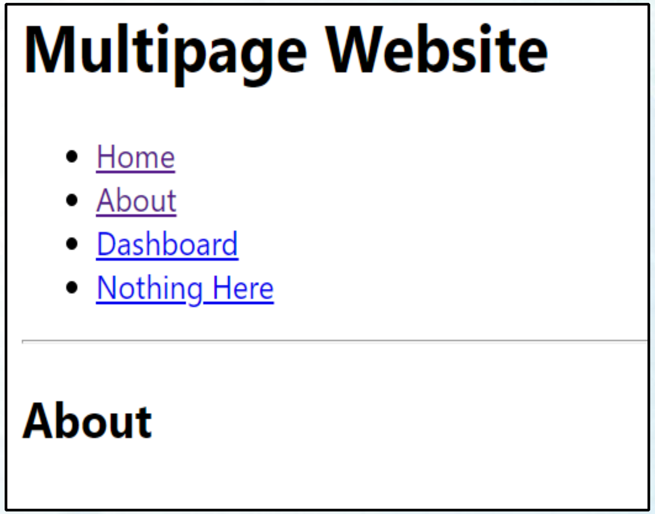
Build a site with core features of React Router including nested routes, outlets, links, and using a "*" route to render a "not found" page.



Multipage Website

- Home
- About
- Dashboard
- Nothing Here

Home

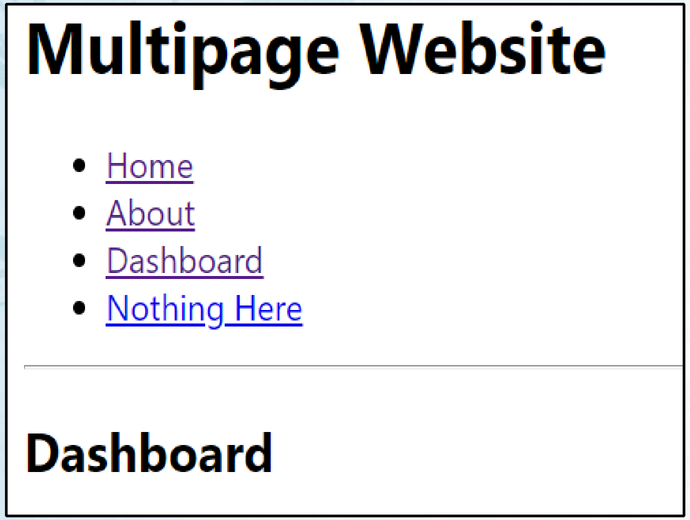


Multipage Website

- Home
- About
- Dashboard
- Nothing Here

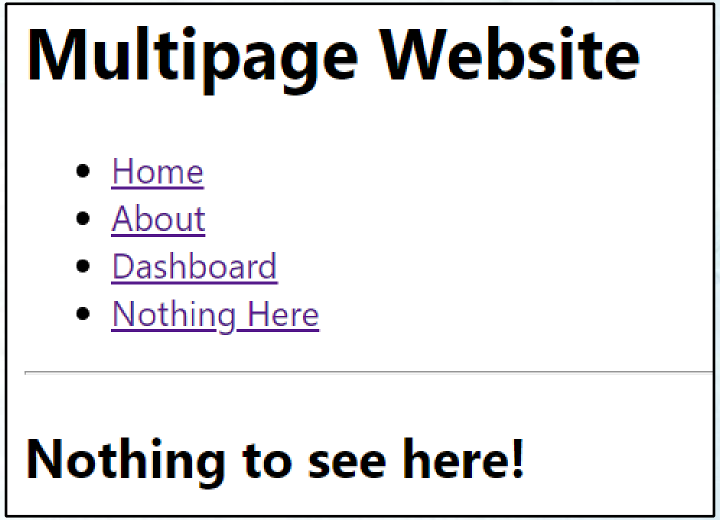
About

6.7 Assignment


Multipage Website

- Home
- About
- Dashboard
- Nothing Here

Dashboard


Multipage Website

- Home
- About
- Dashboard
- Nothing Here

Nothing to see here!