2. React Routing

2.1 What is single and multiple page application and what is routing

- **What is routing in web development?** Routing in web development is mapping of URLs to functions, meaning, it is the process of displaying different content (pages or data) depending on the URL address provided.
    - It is a mechanism where HTTP requests are routed to the code that handles them. Routing comes into play whenever you want to use a URL in your application. Most applications are full of URL links because they're the de facto way of moving around the web. We're used to using physical address systems for navigation, which is like web-based routing with URLs. Put simply, in the Router you determine what should happen when a user visits a certain page.
- **Front-end routing vs back-end routing**: Front-end routing is essentially the same as back-end routing.
- **Routing in front-end development**: front-end routing is a technology for switching between different UI views, based on the changes of the current URL (holding the route). Front-end routing is handled from the browser, The server responses with the defaulted 'index.html' and the browser displays the index.html of the requested URL. Here's where it starts changing from back-end to front-end. When you provide a URL kike "https://www.evangadi.com/class-of-MM-YYYY" on your browser, it triggers an event (an event such as "<a href="/class-of-MM-YYYY">Event A</a>" which written in advance) and then some sort of function runs (that may automatically load data or does something else). **Note**: Front-end routing is more used in single-page applications because single-page applications are basically front-end and back-end separated, and the back-end naturally will not provide routing for the front-end.
    - **Single-page application (SPA)**: SPA is a website/application that loads only a single web document and renders that page on the browser. Single page websites do not need reloading of other pages during use. The way such applications update their body contents is through JavaScript using methods like fetch(). This therefore allows users to use websites without loading whole new pages from the server, which can result in performance gains. **Examples of single page**

applications include: Facebook and Gmail where there is just one web page that you visit which then loads all other content using JavaScript.

   ▪ **Multiple-page applications**: These applications work in a "traditional" way. Every change (such as, display the data or submit data back to server) requests rendering a new page from the server in the browser. These applications are large and due to the amount of content, these applications have many levels of UI. Luckily, it's not a problem anymore. Examples of multi-page applications include eCommerce websites, blogs, forums, other sites that sell products and various services.

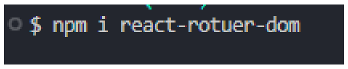
- **Routing in back-end development**: routing is a technology for switching between different server-side endpoints, based on the changes of the requested URL (holding the route). We know that the client sends requests to the server and the server responds back with a response to the client. In back-end routing, the server handles every request by responding to the client with a code 201(I got the thing you want!) or a 404 (Nope, I don't have that!). Below is an example of a simple route where by a text is printed when user enters the "/test" URL path on the browser's navigation bar. That GET request url is sent to the server and the server serves back the requested url to the client as a static file that is stored on the server. If the server doesn't have the requested file, then it returns a 404 response error.

```
● get('/test', function () {
●     return 'Hello World';
● });
```

lets you build single-page applications with multiple views or pages without reloading the browser. Instead of the browser fetching a new HTML page for every route like in the case of multiple page applications, React router handles navigation within your app by changing the URL and rendering different components.

The official site for documentation: https://reactrouter.com/home

To install the package in your react application run the command " npm i react-rotuer-dom" on your terminal. Note that you are in the appropriate location while installing the package.

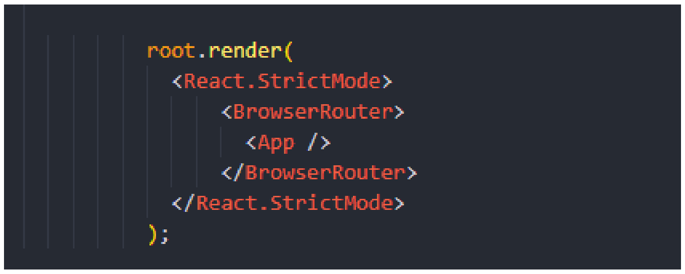


```
$ npm i react-rotuer-dom
```

2.2 Implementation of React routing using react-rotuer-dom

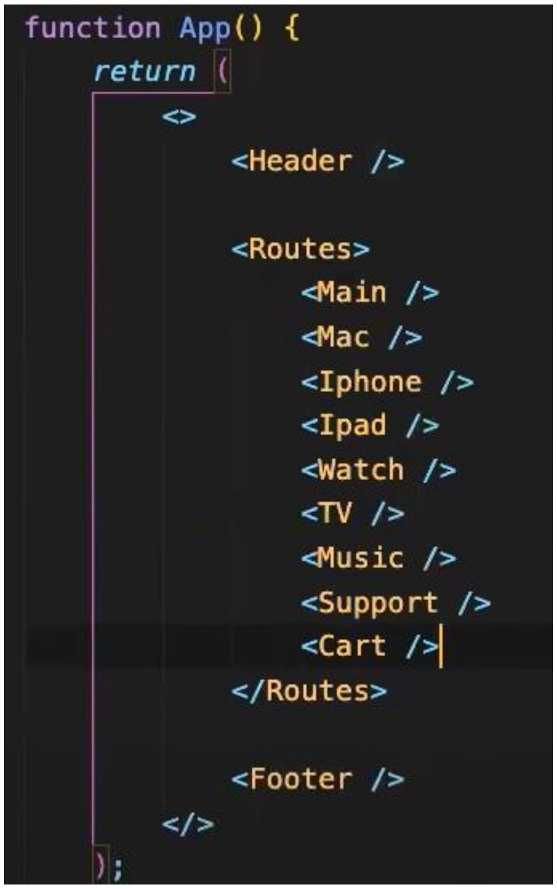
When implementing routing using react-router-dom package , there are four major key terms with functionality which we have to understand.

- BrowserRouter: enables routing in a React application. To set up routing, we wrap our main parent component (in our case, App.js) with the BrowserRotuer component inside the index.js file.



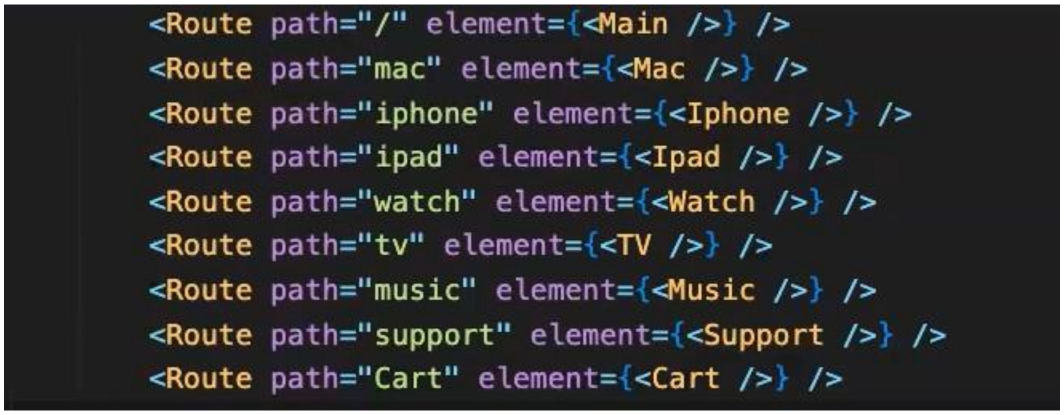
```
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

- Rotues : The Routes component is used to wrap all components that are dependent on routing. When implementing routing, we need to identify which components should render based on a specific route (e.g., pages like Home and About) and which components are not route-dependent (eg., Header and Footer), so they can be placed outside the Rotues component to appear consistently across all pages.



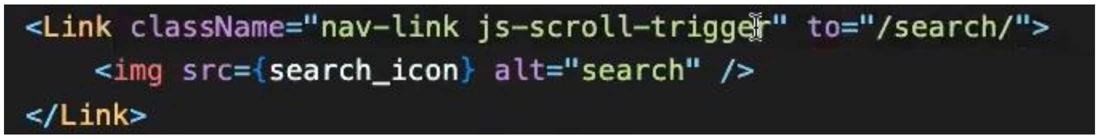
```
function App() {
    return (
        <>
            <Header />

            <Routes>
                <Main />
                <Mac />
                <Iphone />
                <Ipad />
                <Watch />
                <TV />
                <Music />
                <Support />
                <Cart />
            </Routes>

            <Footer />
        </>
    );
```

- Route: is used to define the relationship between a specific URL path and the component that should be rendered when the path is accessed. The Route component has two main properties: path and element. Path (accept string type) defines the URL route where the component should be rendered whereas element (accept component or tags) specifies the component that will be rendered when the corresponding path is accessed.



```
        <Route path="/" element={<Main />} />
        <Route path="mac" element={<Mac />} />
        <Route path="iphone" element={<Iphone />} />
        <Route path="ipad" element={<Ipad />} />
        <Route path="watch" element={<Watch />} />
        <Route path="tv" element={<TV />} />
        <Route path="music" element={<Music />} />
        <Route path="support" element={<Support />} />
        <Route path="Cart" element={<Cart />} />
```

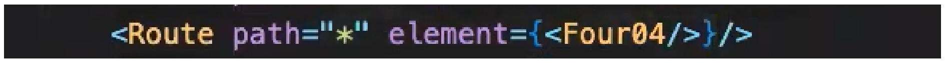
- Link: The Link component from react-router-dom is used to create navigation links between different routes in a React application, similar to an anchor (<a>) tag in HTML. However, unlike an anchor tag, Link does not **reload** the page. In an anchor tag, we use the “href “attribute to define the tag URL, whereas in Link, we use the “to” attribute instead.



```
<Link className="nav-link js-scroll-trigger" to="/search/">
    <img src={search_icon} alt="search" />
</Link>
```

2.3 Non existing route and nested routes

- To handle request that don not match any defined routes in our application, we use a wildcard path (*) in a Rotue by passing it inside path attribute. This is typically used to render a “404 Not Found” message containing component indicating the requested page doesn’t exist.



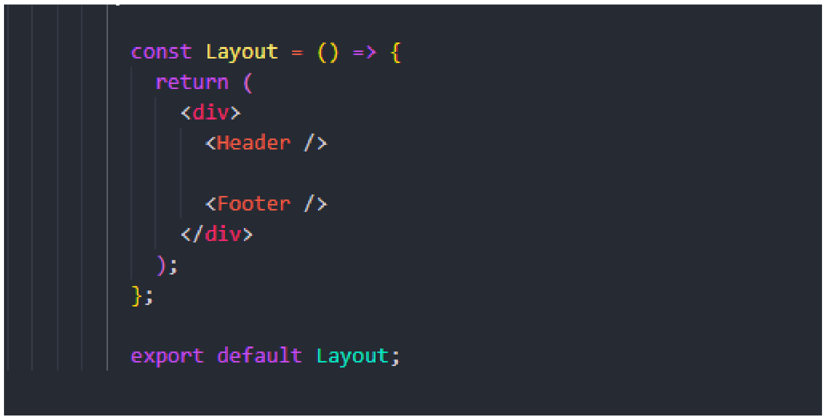
```
      <Route path="*" element={<Four04/>}/>
```

- It is considered a best practice to create Shared components for elements that are used across multiple parts of the application (e.g., Header, Footer, etc). This improves code reusability, allows for more flexible rendering, and enhances overall readability and maintainability of the codebase.

Steps to prepare a shared layout component

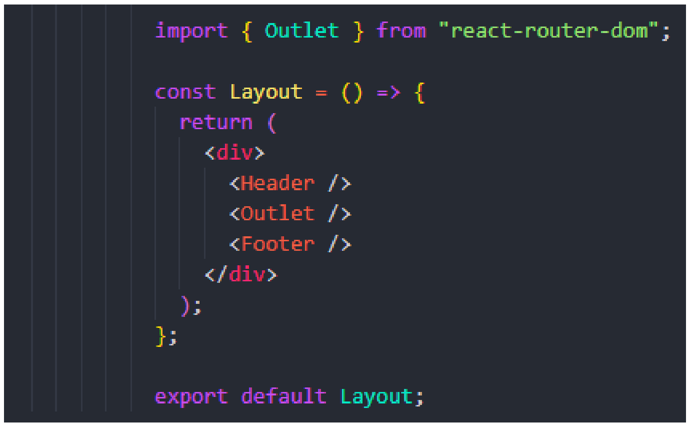
Step 1: Identify teh components that you want to reuse throughout the application (eg. Header and Footer)

Step 2: Create a layout component, and make sure to import and include the shared components you want to use inside it.



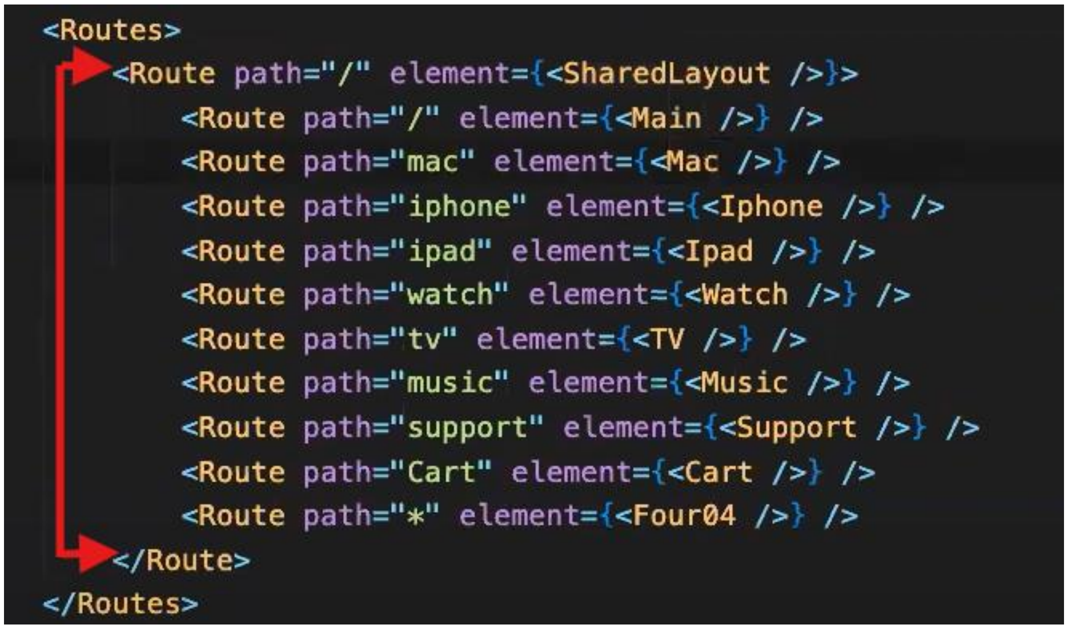
```
const Layout = () => {
  return (
    <div>
      <Header />

      <Footer />
    </div>
  );
};

export default Layout;
```

Step 3: Use the "Outlet" component from react-router-dom to render the nested (child) route components. Place the Outlet in the correct position within your layout component, in our case between the Header and Footer component. The location of Outlet determines where and how the nested components will be displayed, preserving the hierarchy and structure of your application.



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

const Layout = () => {
  return (
    <div>
      <Header />
      <Outlet />
      <Footer />
    </div>
  );
};

export default Layout;
```

Step 4: wrap the components that are identified to be **nested** inside the parent rotue using the Route component structure. This ensures that the nested components will be properly rendered within the Outlet of teh layout component.



```
<Routes>
  <Route path="/" element={<SharedLayout />}>
      <Route path="/" element={<Main />} />
      <Route path="mac" element={<Mac />} />
      <Route path="iphone" element={<Iphone />} />
      <Route path="ipad" element={<Ipad />} />
      <Route path="watch" element={<Watch />} />
      <Route path="tv" element={<TV />} />
      <Route path="music" element={<Music />} />
      <Route path="support" element={<Support />} />
      <Route path="Cart" element={<Cart />} />
      <Route path="*" element={<Four04 />} />
  </Route>
</Routes>
```

2.4 Watch the demo video (2.4) on dynamic routing to understand how to fetch data from the backend and display the fetched data.

2.5 Watch the demo video (2.5) on dynamic routing and dynamic rendering to understand dynamically rendering the content on the frontend based on the fetched data using parameter (e.g productId).

2.6 Watch the additional video (2.6) for tips up on deploying your application and integrating useful libraries that enhance your development experience.

Note :

React Router DOM is frequently updated, and its implementation patterns may change over time. To stay current and follow best practices, we strongly recommend checking the official documentation at "https://reactrouter.com/" whenever you start a new project that involves routing.