

Agenda:

- React Project Using Components
- Routing in React
- Installing and setting up react router
- Link,Index Routes,Nested Routes,Not Found Routes,etc

Sample Project Creation

This one is a basic example of the react application using multiple components. We are going to show you how to create a basic website layout in React. Here we are going to create a basic layout of react application using multiple components like below. We will be using VS Code as our Code Editor.

Way to Create React Application using Multiple Components

1. Create react application
2. Create separate components
3. Add multiple components in single component

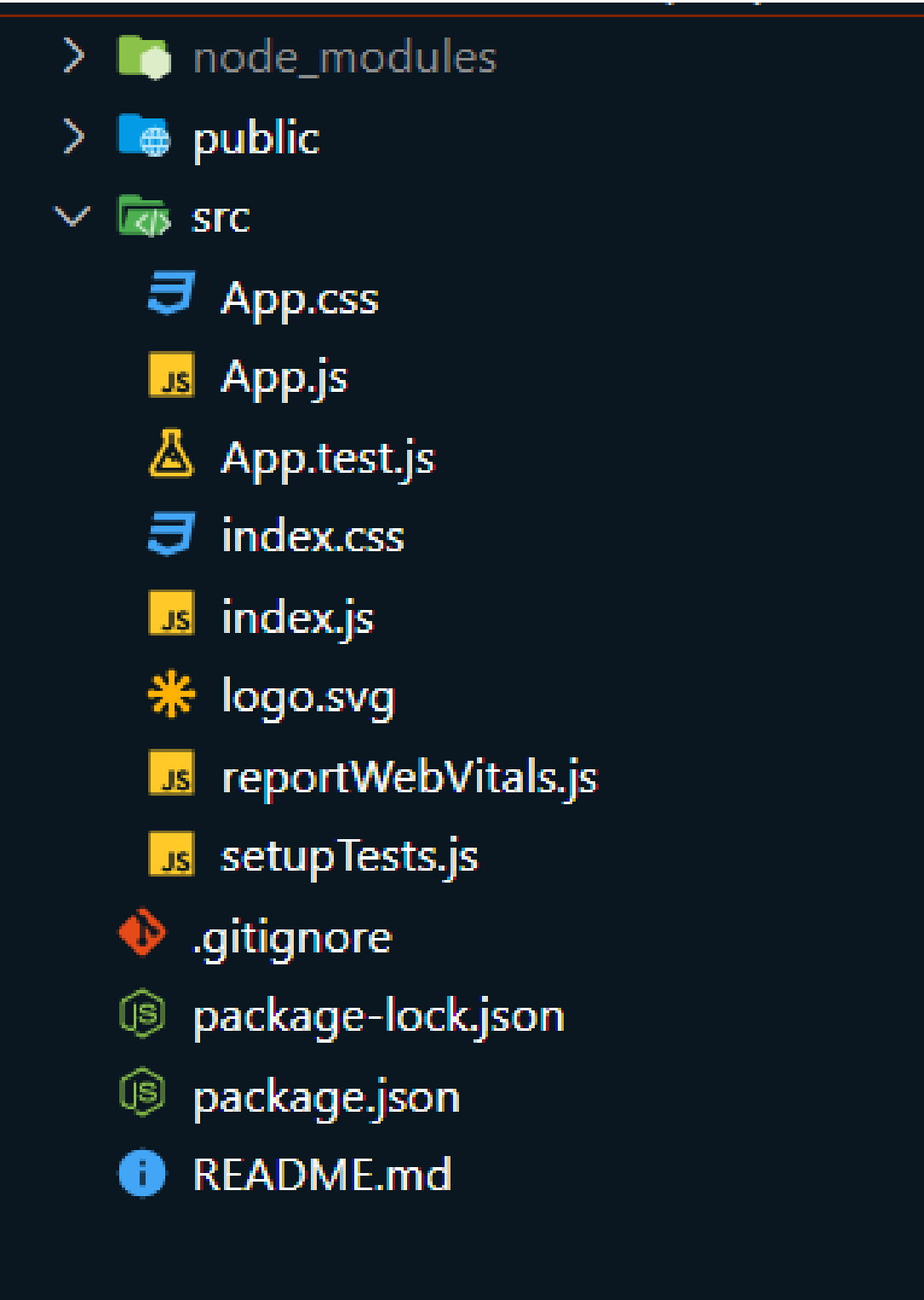
1. Create react application

First, we have to create react application and for that, we used create-react-app npm package. We type the below line in our terminal

```
npx create-react-app react-sample-project
```

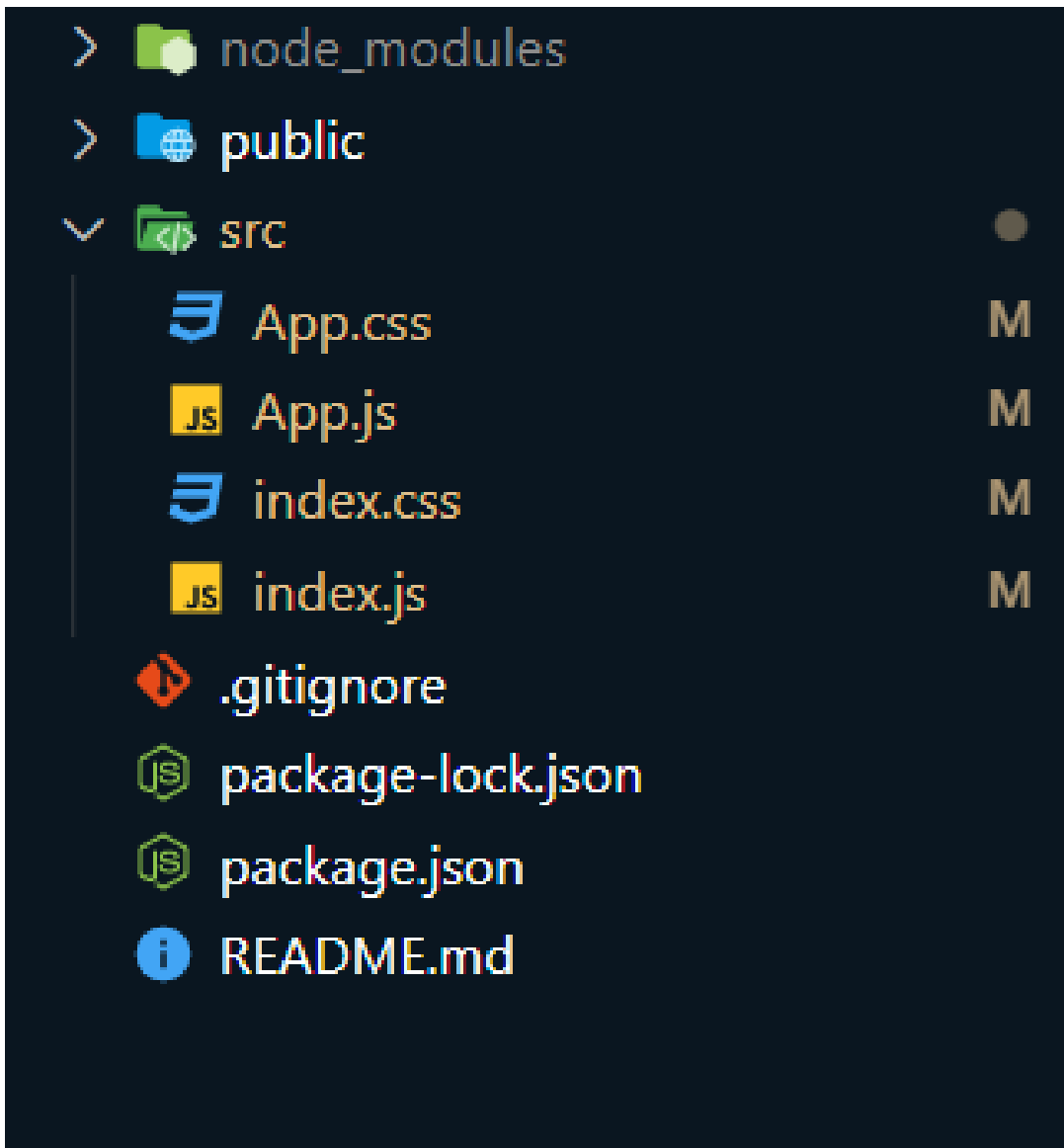


After successful creation of your app your project directory will look something like this



```
> node_modules
> public
▼ src
  App.css
  App.js
  App.test.js
  index.css
  index.js
  logo.svg
  reportWebVitals.js
  setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

We will delete the unwanted files and finally your directory will look something like this



2. Create separate components

We will split our layout into five different components listed below.

- Header Component
- Navbar Component
- Sidebar Component
- Content Component
- Footer Component

Create folder name as "components" to manage the structure of application and we will add all the components inside the folder.

src/components/Header.js

```
import { Component } from "react";

class Header extends Component {
  render() {
    return (
      <div class='p-5 mb-0 bg-light rounded-3'>
        <div class='container-fluid py-5'>
          <h1 class='display-5 fw-bold text-center'>Demo Application</h1>
          <p class='fs-4 text-center'>Demo Application created in React</p>
        </div>
      </div>
    );
  }
}
```



```
    }  
  }  
}
```

```
export default Header;
```

src/components/Navbar.js

```
import { Component } from "react";  
  
class Navbar extends Component {  
  render() {  
    return (  
      <nav class='navbar navbar-expand-lg navbar-dark bg-dark'>  
        <div class='container-fluid'>  
          <a class='navbar-brand' href='#'>  
            Navbar  
          </a>  
          <button  
            class='navbar-toggler'  
            type='button'  
            data-bs-toggle='collapse'  
            data-bs-target='#navbarNav'  
            aria-controls='navbarNav'  
            aria-expanded='false'  
            aria-label='Toggle navigation'  
            <span class='navbar-toggler-icon'></span>  
          </button>  
          <div class='collapse navbar-collapse' id='navbarNav'>  
            <ul class='navbar-nav'>  
              <li class='nav-item'>  
                <a class='nav-link active' aria-current='page' href='#'>  
                  Home  
                </a>  
              </li>  
              <li class='nav-item'>  
                <a class='nav-link active' aria-current='page' href='#'>  
                  About  
                </a>  
              </li>  
              <li class='nav-item'>  
                <a class='nav-link active' aria-current='page' href='#'>  
                  Contact  
                </a>  
              </li>  
            </ul>  
          </div>  
        </div>  
      </nav>  
    );  
  }  
}  
  
export default Navbar;
```

src/components/Sidebar.js

```
const Sidebar = () => {  
  return (  
    <div className='container-fluid bg-light' style={{ height: "100%" }}>  
      <h2>Arco bibendum</h2>  
      <h5>Sit amet mattis vulputate</h5>  
      <p>  
        Non blandit massa enim nec dui nunc mattis enim. Egestas tellus rutrum  
        tellus pellentesque eu tincidunt tortor aliquam nulla..  
      </p>  
    </div>  
  );  
}
```

```

    </p>
    <h3>Massa enim</h3>
    <p>Lorem ipsum dolor sit ame.</p>
  </div>
);
};

export default Sidebar;

```

src/components/Content.js

```

import { Component } from "react";

class Content extends Component {
  render() {
    return (
      <div className='container-fluid bg-primary' style={{ height: "100%" }}>
        <h2>Lorem ipsum dolor</h2>
        <h5>quam pellentesque, Dec 10, 2018</h5>
        <p>Nisi vitae suscipit..</p>
        <p>
          Semper quis lectus nulla at. Nullam ac tortor vitae purus faucibus
          ornare suspendisse. Nunc faucibus a pellentesque sit. Risus quis
          varius quam quisque id diam vel quam elementum. Ornare aenean euismod
          elementum nisi quis eleifend quam.
        </p>
        <h2>Placerat vestibulum</h2>
        <h5>elementum integer enim neque, Sep 21, 2018</h5>
        <p>Bibendum est ultricies..</p>
        <p>
          Semper quis lectus nulla at. Nullam ac tortor vitae purus faucibus
          ornare suspendisse. Nunc faucibus a pellentesque sit. Risus quis
          varius quam quisque id diam vel quam elementum.
        </p>
      </div>
    );
  }
}

export default Content;

```

src/components/Footer.js

```

import React, { Component } from 'react';

class Footer extends Component {
  render() {
    return (
      <div className="footer">
        <h2>Footer</h2>
      </div>
    );
  }
}

export default Footer

```

To apply style for application, we will use bootstrap and also we will add some basic styles in App.css file.

To install bootstrap:

```
npm install bootstrap@5.2.0-beta1
```

To include in index.js import following:

```
import "bootstrap/dist/css/bootstrap.css"; // bootstrap
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.css";

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

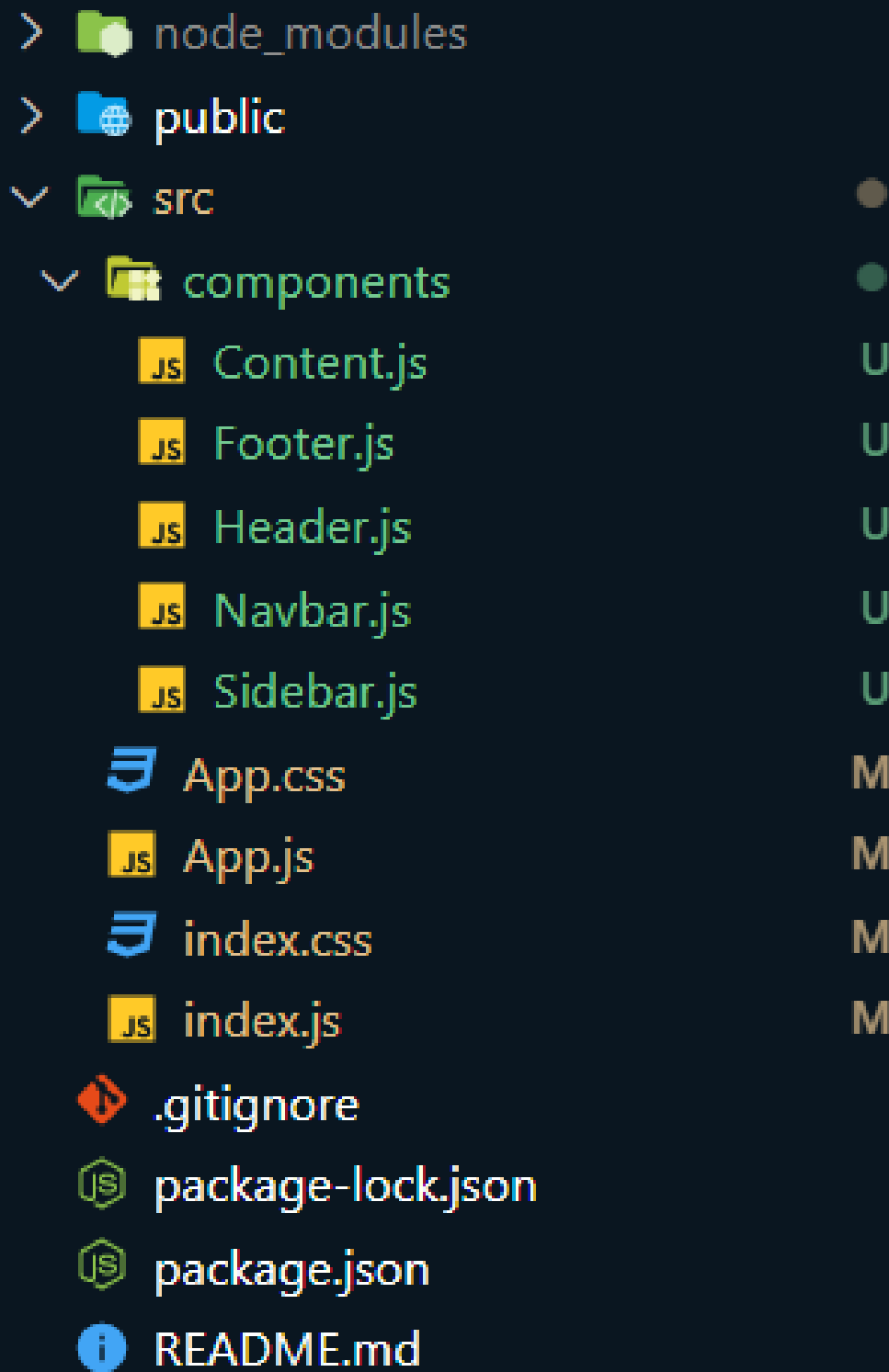
App.css:

```
/*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.row > * {
  padding-right: 0 !important;
  padding-left: 0 !important;
}

/* Footer */
.footer {
  padding: 20px;
  text-align: center;
  background: #ddd;
}
```

After creating the components, your structure will be display look like below.



3. Add multiple components in single component

Import all components in App.js file and add it in below format.

src/App.js

```
import './App.css';
import Content from './components/Content';
import Footer from './components/Footer';
import Header from './components/Header';
import Navbar from './components/Navbar';
import Sidebar from './components/Sidebar';
```



```
function App() {
  return (
    <div>
      <Header />
      <Navbar />
      <div className='row'>
        <Content />
        <Sidebar />
      </div>
      <Footer />
    </div>
  );
}

export default App;
```

Finally run your app using `npm start`

OUTPUT:

![[Screenshot 2022-06-29 125443 (1).png]](https://almablog-media.s3.ap-south-1.amazonaws.com/Screenshot_2022_06_29_125443_1_a46e195bf8.png)

Routing In React

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if the URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

Up until this point, you have dealt with simple projects that do not require transitioning from one view to another, thus, you are yet to interact with Routing in React. In this section, you will get introduced to routing in a React application. To extend your applications by adding routing capabilities, you will use the popular [React-Router](#) library.

React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and is mainly used for developing single page web applications.

Need of React Router

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram use React Router for rendering multiple views.

React Router Installation

It's worth noting that this library has three variants:

=> **react-router**: the core library

=> **react-router-dom**: a variant of the core library meant to be used for web applications

=> **react-router-native**: a variant of the core library used with react native in the development of Android and iOS applications.

Often, there is no need to install the core react-router library by itself, but rather a choice is made between react-router-dom and react-router-native depending on the situation. Both react-router-dom and react-router-native import all the functionality of the core react-router library.

The scope of this module is in the realm of web applications so we can safely choose react-router-dom. This library is installed in a project by running the command below in the project directory.

We will be using the latest version of react-router-dom i.e version 6

```
npm install react-router-dom@6
```



How to Set Up React Router

The first thing to do after installation is complete is to make React Router available anywhere in your app.

To do this, open the `index.js` file in the `src` folder and import `BrowserRouter` from `react-router-dom` and then wrap the root component (the `App` component) in it.

This is what the `index.js` looked like initially:


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

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



After making changes with React Router, this is what you should have:

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

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



All we did was replace `React.StrictMode` with `BrowserRouter` which was imported from `react-router-dom`. Now the router features are accessible from any part of your app.

How to Route to Other Components

We are finally done setting things up, so now we'll look at routing to and rendering different components.

Step 1 - Create multiple components

We'll create the following `Home`, `About`, and `Contact` components like this:

```
function Home() {
  return (
    <div>
      <h1>This is the home page</h1>
    </div>
  );
}

export default Home;
```



```
import React from 'react'

function About() {
  return (
    <div>
      <h1>This is the about page</h1>
    </div>
  )
}

export default About
```



```
import React from 'react'
```



```
function Contact() {
  return (
    <div>
      <h1>This is the contact page</h1>
    </div>
  )
}

export default Contact
```

Step 2 - Define routes

Since the `App` component acts as the root component where our React code gets rendered from initially, we will be creating all our routes in it. Don't worry if this does not make much sense – you'll understand better after looking at the example below.

```
import { Routes, Route } from "react-router-dom"
import Home from "./Home"
import About from "./About"
import Contact from "./Contact"

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={ <Home/> } />
        <Route path="about" element={ <About/> } />
        <Route path="contact" element={ <Contact/> } />
      </Routes>
    </div>
  )
}

export default App
```

We first imported the features we'll be using – `Routes` and `Route`. After that, we imported all the components we needed to attach a route to. No let's break down the process.

`Routes` acts as a container/parent for all the individual routes that will be created in our app.

`Route` is used to create a single route. It takes in two attributes:

- `path`, which specifies the URL path of the desired component. You can call this pathname whatever you want. Above, you'll notice that the first pathname is a backslash (/). Any component whose pathname is a backslash will get rendered first whenever the app loads for the first time. This implies that the `Home` component will be the first component to get rendered.
- `element`, which specifies the component the route should render.

All we have done now is define our routes and their paths, and attach them to their respective components.

Step 3 - Use `Link` to navigate to routes

If you have been coding along up to this point without any errors, your browser should be rendering the `Home` component.

We will now use a different React Router feature to navigate to other pages based on those routes and pathnames we created in the `App` componer That is:

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

function Home() {
  return (
    <div>
      <h1>This is the home page</h1>
      <Link to="about">Click to view our about page</Link>
      <Link to="contact">Click to view our contact page</Link>
    </div>
  );
}
```

```
export default Home;
```

The `Link` component is similar to the anchor element (`<a>`) in HTML. Its `to` attribute specifies which path the link takes you to.

Recall that we created the pathnames listed in the `App` component so when you click on the link, it will look through your routes and render the component with the corresponding pathname.

Always remember to import `Link` from `react-router-dom` before using it.

Nested Routes

This is one of the most powerful features of React Router making it so you don't have to mess around with complicated layout code. The vast majority of your layouts are coupled to segments of the URL and React Router embraces this fully.

Routes can be nested inside one another, and their paths will nest too (child inheriting the parent).

```
function App() {  
  return (  
    <Routes>  
      <Route path="invoices" element={<Invoices />}>  
        <Route path=":invoiceId" element={<Invoice /> } />  
        <Route path="sent" element={<SentInvoices /> } />  
      </Route>  
    </Routes>  
  );  
}
```

This route config defined three route paths:

- `"/invoices"`
- `"/invoices/sent"`
- `"/invoices/:invoiceId"`

When the URL is `"/invoices/sent"` the component tree will be:

```
<App>  
  <Invoices>  
    <SentInvoices />  
  </Invoices>  
</App>
```

When the URL is `"/invoices/123"` , the component tree will be:

```
<App>  
  <Invoices>  
    <Invoice />  
  </Invoices>  
</App>
```

In previous versions of React Router you had to order your routes a certain way to get the right one to render when multiple routes matched an ambiguous URL. V6 is a lot smarter and will pick the most specific match so you don't have to worry about that anymore. For example, the URL `/invoices/sent` matches both of these routes:

```
<Route path="invoices/:invoiceId" element={<Team /> } />  
<Route path="invoices/sent" element={<NewTeamForm /> } />
```

But `invoices/sent` is a more specific match than `invoices/:invoiceId` , so `<SentInvoices />` will render.

Notice the inner component that changed with the URL (`<SentInvoices>` and `<Invoice>`). The parent route (`<Invoices>`) is responsible for making sure the matching child route is rendered with `<Outlet>` . Here's the full example:

```
import { Routes, Route, Outlet } from "react-router-dom";  
  
function App() {
```

```

return (
  <Routes>
    <Route path="invoices" element={<Invoices />}>
      <Route path=":invoiceId" element={<Invoice />} />
      <Route path="sent" element={<SentInvoices />} />
    </Route>
  </Routes>
);
}

function Invoices() {
  return (
    <div>
      <h1>Invoices</h1>
      <Outlet />
    </div>
  );
}

function Invoice() {
  let { invoiceId } = useParams();
  return <h1>Invoice {invoiceId}</h1>;
}

function SentInvoices() {
  return <h1>Sent Invoices</h1>;
}

```

<Outlet> :

An **<Outlet>** should be used in parent route elements to render their child route elements. This allows nested UI to show up when child routes are rendered. If the parent route matched exactly, it will render a child index route or nothing if there is no index route.

For Example:

```

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

      /* This element will render either <DashboardMessages> when the URL is
       "/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>
  );
}

```



Index Routes

Index routes can be thought of as "default child routes". When a parent route has multiple children, but the URL is just at the parent's path, you probably want to render something into the outlet.

Consider this example:

```
function App() {  
  return (  
    <Routes>  
      <Route path="/" element={<Layout />}>  
        <Route path="invoices" element={<Invoices />} />  
        <Route path="activity" element={<Activity />} />  
      </Route>  
    </Routes>  
  );  
}  
  
function Layout() {  
  return (  
    <div>  
      <GlobalNav />  
      <main>  
        <Outlet />  
      </main>  
    </div>  
  );  
}
```

This page looks great at "/invoices" and "/activity", but at "/" it's just a blank page in `<main>` because there is no child route to render there. For this we can add an index route:

```
function App() {  
  return (  
    <Routes>  
      <Route path="/" element={<Layout />}>  
        <Route index element={<Activity />} />  
        <Route path="invoices" element={<Invoices />} />  
        <Route path="activity" element={<Activity />} />  
      </Route>  
    </Routes>  
  );  
}
```

Now at "/" the `<Activity>` element will render inside the outlet.

You can have an index route at any level of the route hierarchy that will render when the parent matches but none of its other children do.

```
function App() {  
  return (  
    <Routes>  
      <Route index element={<Home />} />  
      <Route path="dashboard" element={<Dashboard />}>  
        <Route index element={<DashboardHome />} />  
        <Route  
          path="invoices"  
          element={<DashboardInvoices />}  
        />  
      </Route>  
    </Routes>  
  );  
}
```

Relative Links

Relative `<Link to>` values (that do not begin with a `/`) are relative to the path of the route that rendered them. The two links below will link to `/dashboard/invoices` and `/dashboard/team` because they're rendered inside of `<Dashboard>`. This is really nice when you change parent's URL or re-arrange your components because all of your links automatically update.

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

function Home() {
  return <h1>Home</h1>;
}

function Dashboard() {
  return (
    <div>
      <h1>Dashboard</h1>
      <nav>
        <Link to="invoices">Invoices</Link>{" "}
        <Link to="team">Team</Link>
      </nav>
      <hr />
      <Outlet />
    </div>
  );
}

function Invoices() {
  return <h1>Invoices</h1>;
}

function Team() {
  return <h1>Team</h1>;
}

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="dashboard" element={<Dashboard />}>
        <Route path="invoices" element={<Invoices />} />
        <Route path="team" element={<Team />} />
      </Route>
    </Routes>
  );
}
```

“Not Found” Routes

When no other route matches the URL, you can render a "not found" route using `path="*"` . This route will match any URL, but will have the weakest precedence so the router will only pick it if no other routes match.

```
function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="dashboard" element={<Dashboard />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  );
}
```

Multiple Set Of Routes

Although you should only ever have a single `<Router>` in an app, you may have as many `<Routes>` as you need, wherever you need them. Each `<Routes>` element operates independently of the others and picks a child route to render.

```
function App() {  
  return (  
    <div>  
      <Sidebar>  
        <Routes>  
          <Route path="/" element={<MainNav />} />  
          <Route  
            path="dashboard"  
            element={<DashboardNav />}  
          />  
        </Routes>  
      </Sidebar>  
  
      <MainContent>  
        <Routes>  
          <Route path="/" element={<Home />}>  
            <Route path="about" element={<About />} />  
            <Route path="support" element={<Support />} />  
          </Route>  
          <Route path="dashboard" element={<Dashboard />}>  
            <Route path="invoices" element={<Invoices />} />  
            <Route path="team" element={<Team />} />  
          </Route>  
          <Route path="*" element={<NotFound />} />  
        </Routes>  
      </MainContent>  
    </div>  
  );  
}
```

Conclusion

So in this module we created a sample project using multiple components in React and learned in depth about routing in React.

Thank You !