ROUTING

- Finding the path of different pages of our app.
- For that we use a library or npm package called **React Router**.
- Install this package

```
npm i react-router-dom
```

- Make an 'About.js & Contact.js' pages.

To click the 'About' on homepage and to move to the 'About' page, we first have to create a routing configuration.

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

This is a function we get from react-router-dom. It will help us create routing.

→ To create the routing, do :-

```
const appRouter = createBrowserRouter([
  {
    path : "/",
    element : <AppLayout/>,
  },
  {
    path : "/about",
    element : <About/>
  },
]);
```

Here, I will define where My app will get loaded with "/path". This is where we create Routing configuration

[→ There are multiple Routers. Read Doc]

"createBrowserRouter" is the most recommended route for all React Router web projects.

→ This alone won't work, we need to provide this appRouter to our app.

For that there is a component RouterProvider which is coming from react-router dom. Import it.

→ And render this RouterProvider in your App.js.

```
root.render(<RouterProvider router={appRouter} />);
```

→ 'react-router-dom' is a powerful library which ~~shows~~ gives us a better UI for showing us this error pages.

Create an error page of your own "Error.js" and pass this component to our router config.

```
{
  path: "/",
  element: <AppLayout />,
  errorElement: <Error />,
}
```

If there is an error in the path, it will load the error element.

→ To show more information about the error in the errorpage, 'react-router-dom' gives us {useRouteError}. Import this in Error.js.

```
import {useRouteError} from "react-router-dom"
```

→ { useRouteError } is a hook : which won't allow that red-colour error to come in consoles. It catch all the routing errors and we can show those error to the user.

Error.js

```
import { useRouteError } from "react-router-dom";
const Error = () => {
    Const err = useRouteError();
    const {
    return (
      <div>
        <h1>OOps! Something went wrong</h1>
        <h2>
          {err.status + ":" + err.statusText}
        </h2>
      </div>
    );
};
export default Error;
```

⇒ Problem with anchor tag

It will reload the entire page when it is clicked. It disrupts user experience and can result in a slower page load time. This cause problems for single-page applications (SPAs)

Single Page Applications (SPA)

- React apps are SPAs
- Having SPAs will not reload, It will not make network call when we are changing pages
- Loads a single HTML page and dynamically update the page in response to user interaction without reloading the entire page.
- This approach allows for faster navigation and a more seamless user experience.

Two types of Routing

1. Client-side routing
2. Server-side routing

Server-Side Routing

- all our pages come from server.
- make a network call, get the html, js, css and loads the whole page.

Client-side Routing

→ dynamically update content of SPA in response to change in URL.

→ don't do full page reload.

→ 'React-router-dom' gives {Link}.

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

<Link to = "/about" >
    <li> About </li>
</Link>
```

→ To keep Header & Footer stick on to every page, change the routing config.
ie, to make the about page children of <AppLayout /> do:

```
Const appRouter = createBrowserRouter([
  {
    path : "/"
    element : <AppLayout/>,
    errorelement : <Error/>
    children : [
      { path : "/about",
        element : <About/>,
      }, ] }, ]);
```

```
const AppLayout = () => {
    return (
        <>
            <Header />
            <Outlet />
            <Footer />
        </>
    );
};
```

→ React-router-dom gives acess to Outlet.
This outlet will be filled by the configuration.
So, all the children will go inside Outlet.
acording to the route.

DYNAMIC ROUTING

→ process of rendering components in response to a change in the application's URL.

→ If the route to restaurant menu page is like

```
{
    path : "/restaurant/:id"
    element : <RestMenu />
}
```

id is dynamic (it can be anything).

→ To read the 'id' passed in URL,
'react-router-dom' gives us { useParams }.
It's the routing parameters.

<RestMenu/> →

→ import { useParams } from "react-router-dom";

```
const RestMenu =() => {
    const params = useParams();
    const {id} = params;
    return (
      <div>
        <h1>Restaurant id : 1234</h1>
          <h2>Namaste</h2>
      </div>
    );};
export default RestMenu;
```

{id}

→ We will be having the id inside params.