

Episode-07 | Finding The Path



Please make sure to follow along with the whole "Namaste React" series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React development.




I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-07** first. Understanding what "Akshay" shares in the video will make these notes way easier to understand.

Q) What are various ways to add images to our App? Explain with code examples

There are several ways to add and display images.

1. Importing images using ES6 Modules
2. Using public folder
3. Loading images from a remote source
4. Using image assets within CSS

1 `Importing images using ES6 Modules`  We can import images directly using ES6 modules. This is a common approach for `small to medium-sized apps`, and it's straightforward. Firstly, We have to place our image in the project directory (e.g., in the src folder or a subfolder).

Example:

```
import React from 'react';
import myImage from './my_image.jpg';

function App() {
  return (
    <div>
      <img src={myImage} alt="My Image" />
    </div>
  );
}

export default App;
```

2 **Using public folder** ⓘ If we want to reference images in the public folder, we can do so without importing them explicitly. This method is useful for handling large image assets or for dynamic image URLs. Place your image in the public directory.

```
// public/my_image.jpg
```

Then, reference it in your code:

```
import React from 'react';

function App() {
  return (
    <div>
      <img src={process.env.PUBLIC_URL + '/my_image.jpg'} alt="My Image" />
    </div>
  );
}

export default App;
```

3 **Loading images from a remote source** ⓘ We can load images from a remote source,

such as an external URL or a backend API, by specifying the image URL directly in our img tag.

Example:

```
import React from 'react';

function App() {
  const imageUrl = 'https://example.com/my_image.jpg';

  return (
    <div>
      <img src={imageUrl} alt="My Image" />
    </div>
  );
}

export default App;
```

4 **Using image assets within CSS** ? We can also use images as background images or in other CSS styling. In this case, we can reference the image in your CSS file.

Example CSS (styles.css):

```
.image-container {
  background-image: url('/my_image.jpg');
  width: 300px;
  height: 200px;
}
```

Then, apply the CSS class to your JSX?

```
import React from 'react';
import './styles.css';

function App() {
  return (
    <div className="image-container">
      {/* Content goes here */}
    </div>
  );
}

export default App;
```

Choose the method that best fits your project's requirements and organization. Importing images using ES6 modules is the most common and convenient approach for most React applications, especially for small to medium-sized projects. For larger projects with many images, consider the folder structure and organization to keep our code clean and maintainable.

Q) What would happen if we do `console.log(useState())`?

If you use `console.log(useState())` in a React functional component, it will display the result of calling the `useState()`

function in our browser's developer console. The `useState()` function is a React Hook that is typically used to declare a state variable in a functional component. When we call `useState()`, it returns an array with two elements: the current state value and a function to update that state value .

For example:

```
const [count, setCount] = useState(0);
```

In this example, **count** is the current state value, and **setCount** is the function to update it.

If we do `console.log(useState())`, we will see something like this in the console:

```
[0, Function]
```

The first element of the array is the initial state value (in this case, `0`), and the second element is the function to update the state. However, using `console.log(useState())` directly in our component without destructuring the array and assigning names to these elements isn't a common or recommended practice. Normally, we would destructure the array elements when using `useState()` to make our code more readable and maintainable.

So, it's more typical to use `useState()` like this:

```
const [count, setCount] = useState(0);  
console.log(count); // Logs the current state value  
console.log(setCount); // Logs the state update function
```

This way, we can access and work with the state and state update function in our component.

Q) How will `useEffect` behave if we don't add a dependency array?

In React, when we use the `useEffect` hook without providing a dependency array, the effect will be executed on every render of the component. This means that the code inside the `useEffect` will run both after the initial render and after every subsequent render.



Here's an example of using `useEffect` without a dependency array

```
import React, { useEffect } from 'react';

function MyComponent() {
  useEffect(() => {
    // This code will run on every render
    console.log('Effect executed');
  });

  return (
    <div>
      {/* Component content */}
    </div>
  );
}
```

In this example, the `useEffect` without a dependency array doesn't specify any dependencies, so it will run after every render of `MyComponent`. This behavior can be useful in some cases, but it's essential to be cautious when using `useEffect` without a dependency array because it can lead to performance issues, especially if the effect contains expensive operations.

When we don't provide a dependency array, the effect is considered to have an empty dependency array, which is equivalent to specifying every value as a dependency. Therefore, it's important to understand the consequences of running the effect on every render and to use this pattern judiciously.

In many cases, we might want to include a dependency array to control when the effect should run based on changes in specific variables or props. This can help optimize the performance of our component and prevent unnecessary re-renders.



```
useEffect(() => {}, []);
```

Case 1, when the dependency array is not included as an argument in the `useEffect` hook, the

callback function inside `useEffect` will be executed every time the component is initially rendered and subsequently re-rendered. This means that the effect runs on every render cycle, and there are no dependencies that control when it should or should not execute.

Here's the relevant code again for reference:

```
import React, { useEffect } from 'react';

function MyComponent() {
  useEffect(() => {
    // This code will run on every render
    console.log('Effect executed');
  });

  return (
    <div>
      {/* Component content */}
    </div>
  );
}
```

The callback function in the `useEffect` will log `Effect executed` to the console every time `MyComponent` is rendered or re-rendered. This behavior can be useful in some cases but should be used carefully to avoid excessive or unnecessary executions of the effect. If we want more control over when the effect should run, we can include a dependency array to specify the dependencies that trigger the effect when they change.

In Case 2, when the dependency array is empty (i.e., `[]`) in the arguments of the `useEffect` hook, the callback function will indeed be executed once during the initial render of the component. However, it won't be limited to the initial render only. It will run after the initial render and then on every re-render of the component.

Here's an example of using `useEffect` with an empty dependency array:

```
import React, { useEffect } from 'react';

function MyComponent() {
  useEffect(() => {
    // This code will run after the initial render and on every re-render
    console.log('Effect executed');
  }, []);

  return (
    <div>
      {/* Component content */}
    </div>
  );
}
```

That's not accurate. In Case 2, when the dependency array is empty (i.e., []) in the arguments of the `useEffect` hook, the callback function will indeed be executed once during the initial render of the component. However, it won't be limited to the initial render only. It will run after the initial render and then on every re-render of the component.

Here's an example of using `useEffect` with an empty dependency array:

```
import React, { useEffect } from 'react';

function MyComponent() {
```



```

useEffect(() => {
  // This code will run after the initial render and on every re-render
  console.log('Effect executed');
}, []);

return (
  <div>
    {/* Component content */}
  </div>
);
}

```

In this case, the callback function in the `useEffect` with an empty dependency array will run once after the initial render and then on every subsequent re-render of `MyComponent`. It won't run if the component is unmounted and then remounted, but it will run whenever the component is re-rendered, even if there are no dependencies to watch for changes.

If you want the effect to run only once, and not re-run on re-renders, you can specify an empty dependency array like this:

```

useEffect(() => {
  // This code will run only once, after the initial render
  console.log('Effect executed');
}, []);

```

In this case, the effect will run only after the initial render, and it won't run again on subsequent re-renders.

Case 3 - When the dependency array in the arguments of the `useEffect` hook contains a condition (a variable or set of variables), the callback function will be executed once during the initial render of the component and also on re-renders if there is a change in the condition.

Here's an example of using `useEffect` with a condition in the dependency array :

```
import React, { useEffect, useState } from 'react';

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

  useEffect(() => {
    // This code will run after the initial render and whenever 'count' changes
    console.log('Effect executed');
  }, [count]);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Increment Count</button>
      <p>Count: {count}</p>
    </div>
  );
}
```

In this case, the `useEffect` has `count` as a dependency in the array. This means that the effect will run after the initial render and then again whenever the `count` variable changes. If we click the Increment Count button, the `count` state will change, triggering the effect to run again. If the condition specified in the dependency array doesn't change, the effect won't run on re-renders.

This allows us to control when the effect runs based on specific conditions or dependencies. It's a useful way to ensure that the effect only runs when the relevant data or state has changed.

Q) What is SPA ?

SPA stands for `Single Page Application` . It's a type of web application or website that interacts with the user by

dynamically rewriting the current web page rather than loading entire new pages from the server. In other words, a single HTML page is loaded

initially, and then the content is updated dynamically as the user interacts with the application, typically through JavaScript.

Key characteristics of SPAs include :

Dynamic Updates ? In SPAs, content is loaded and updated without requiring a full page reload. This is achieved using JavaScript and client-side routing.

Smooth User Experience ? SPAs can provide a smoother and more responsive user experience because they can update parts of the page without the entire page needing to be refreshed.

Faster Initial Load ? While the initial load of an SPA might take longer as it downloads more JavaScript and assets, subsequent interactions with the application can be faster because only data is exchanged with the server and not entire HTML pages.

Client-Side Routing ? SPAs often use client-side routing to simulate traditional page navigation while staying on the same HTML page. This is typically achieved using libraries like React Router or Vue Router.

API-Centric ? SPAs are often designed to be more API-centric, where the client communicates with a backend API to fetch and send data, usually in JSON format. This allows for decoupling the front end and back end.

State Management ? SPAs often use state management libraries (e.g., Redux for React or Vuex for Vue) to manage the application's state and data flow.

Popular JavaScript frameworks and libraries like React, Angular, and Vue are commonly used to build SPAs. They offer tools and patterns to create efficient and maintainable single-page applications.

Q) What is the difference between Client Side Routing and Server Side Routing ?

A ? Client-side routing and server-side routing are two different approaches to handling routing and navigation in web applications. They have distinct characteristics and are often used for different purposes. Here's an overview of the key differences between them:

Client-Side Routing : **Handling on the Client** ? In client-side routing, routing and navigation are managed on the client side, typically within the web browser. JavaScript frameworks and libraries, such as React Router (for React applications) or Vue Router (for Vue.js applications), are commonly used to implement clientside routing.

Faster Transitions ? Client-side routing allows for faster page transitions since it doesn't require the server to send a new HTML page for each route change. Instead, it updates the DOM and URL dynamically without full page reloads.

Single-Page Application (SPA) ? Client-side routing is often associated with single-page applications ?SPAs), where the initial HTML page is loaded, and subsequent page changes are made by updating the content using JavaScript.

SEO Challenges ? SPAs can face challenges with search engine optimization ?SEO? because search engine crawlers may not fully index the content that relies heavily on client-side rendering. Special techniques like server-side rendering ?SSR? or pre-rendering can be used to address this issue.

Route Management ? Routing configuration is typically defined in code and managed on the client side, allowing for dynamic and flexible route handling.

- - **Server-Side Routing** :

Handling on the Server ? Server-side routing manages routing and navigation on the server. When a user requests a different URL, the server generates and sends a new HTML page for that route.

Slower Transitions ? Server-side routing tends to be slower in terms of page transitions compared to client-side routing, as it involves full page reloads.

Traditional Websites ? Server-side routing is commonly used for traditional multipage websites where each page is a separate HTML document generated by the server.

SEO-Friendly ? Server-side routing is inherently more SEO-friendly, as each page is a separate HTML document that can be easily crawled and indexed by search engines.

Route Configuration ? Routing configuration in server-side routing is typically managed on the server, and URLs directly correspond to individual HTML files or routes.

In summary, client-side routing is suitable for building SPAs and offers faster, more interactive user experiences but can pose SEO challenges. Server-side routing is more SEO-friendly and is

used for traditional websites with separate HTML pages, but it can be slower in terms of page transitions. The choice between these two routing approaches depends on the specific requirements and goals of a web application or website. In some cases, a hybrid approach that combines both client-side and server-side routing techniques may be used to achieve the best of both worlds.

Self-Notes

We are going to create different pages or different url like localhost1234/about so It take us the about page, if we do /contact it should take us the contact page and so on. We will be using Routing library in this episode.

Dive deeper in useEffect hook

How to useEffect hook call-

```
import {useEffect } from 'react';
```

```
useEffect(() => {  
    fetchData();  
}, []);
```

useEffect will called using two arguments

1. Callback function (Mandatory)
2. Dependency arrays (Not Mandatory)

When the useEffect will call – It will called after every render of the component
Means useEffect will be called every time my component renders useEffect will be called, every time my header component will be rendered my useEffect will be called.
But because we have put dependency array over here so this dependency array changes the behaviour of its render.

Without the dependency array it will render every time my component renders

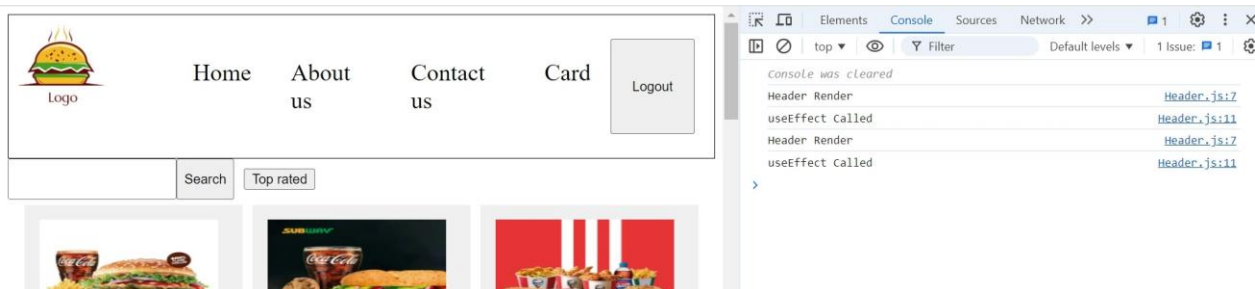
If no dependency array => useEffect is called on every render

useEffect will be called every time my header(component) rendered

```
// named export
export const Header = () => {
  const [btnNameReact, setBtnNameReact] = useState("Login")
  console.log("Header Render")

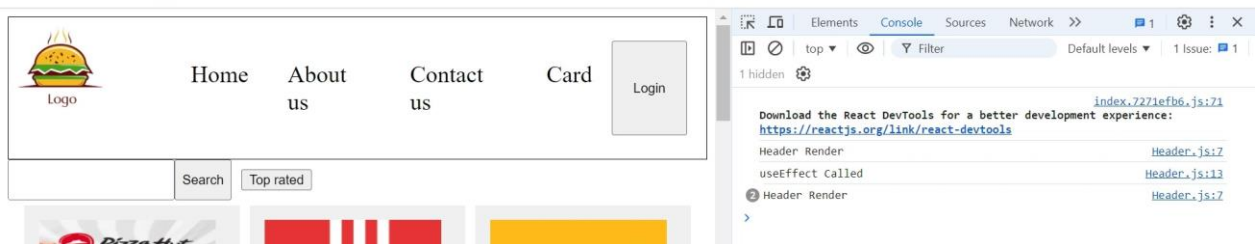
  // If no dependency array => useEffect is called on every render
  useEffect(() => {
    console.log("useEffect Called")
  });

  return (
    <div>
      <div>
        <img alt="Logo" data-bbox="132 341 181 378"/>
        Home About us Contact us Card Logout
      </div>
      <div>
        Search Top rated
      </div>
      <div>
        <img alt="Coca-Cola" data-bbox="148 428 248 468"/>
        <img alt="Coca-Cola" data-bbox="278 428 391 468"/>
        <img alt="Coca-Cola" data-bbox="418 428 541 468"/>
      </div>
    </div>
  )
}
```



Even after clicking of login logout button header will call and useEffect will call in every render. When the dependency array is not present.

If dependency array is empty = [] => useEffect is called on initial render (just once)



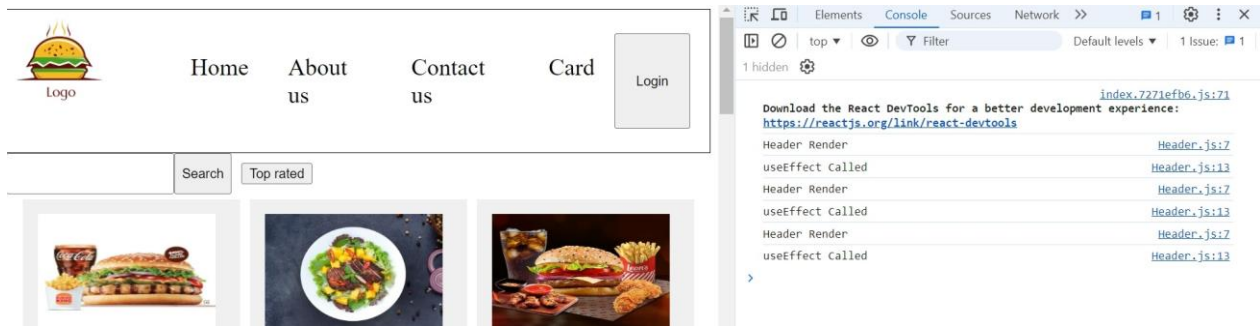
When the page loads Header rendered useEffect is called , but useEffect is called only once, only initial rendered when the page loads, component loads useEffect is called . It not be called again and again even if the component re-renders.

If I click my login logout button the count of header render getting increase but useEffect not getting called again and again.

If dependency array is [btnNameReact] => called everytime btnNameReact is updated.

Everytime my btnNameReact changes my useEffect will be called.

It will call in initial render. If the dependency btnNameReact so whenever my btnNameReact changes from login to logout useEffect will be called



Dive deeper in useState hook

- Whenever using useState never use useState outside of your component otherwise it will through an error. So always use inside the body.
- useState has specific purpose it is used to create the local state variable inside your function component. So, always calls inside functional component and try to call this hook on the top, means when the functions starts always try to use useState on the top so you don't have inconsistency in your code. JS is the synchronous single threaded language the code will run line by line so when you start from the top so first think you do is to create the state variable. It the good practice and the habits. Also react understand properly.
- Never use useState inside the if else condition. JavaScript allow to do this but don't do this, it create the inconsistency in your program

```
if () {  
  const [searchText, setSearchText] = useState("");  
}
```

- Don't create state variable inside the function

```
const Body = () => {
  const [listOfRestaurants, setListOfRestraunt] = useState([]);
  const [filteredRestaurant, setFilteredRestaurant] = useState([]);

  function () {
    const [searchText, setSearchText] = useState("");
  }
}
```

The state variables are meant to be created inside the functional component on the higher level. Try to keep it on the top.

Routing-

Install react routing library –

Website - <https://reactrouter.com/en/main>

Command – npm i react-router-dom

```
PS C:\Users\PKHARD\Desktop\NamasteReact> npm i react-router-dom

added 3 packages, and audited 198 packages in 8s

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

found 0 vulnerabilities
npm notice
npm notice New major version of npm available! 9.2.0 -> 10.9.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.9.0
npm notice Run npm install -g npm@10.9.0 to update!
eact> npm start

> namastereact@1.0.0 start
> parcel index.html
```



```

10 package.json > {} devDependencies
11   "repository": {
12     "url": "git+https://github.com/pragkhard/namaste-reactjs.git"
13   },
14   "keywords": [
15     "namaste",
16     "react",
17     "react"
18   ],
19   "author": "Pragati khard",
20   "license": "ISC",
21   "bugs": {
22     "url": "https://github.com/pragkhard/namaste-reactjs/issues"
23   },
24   "homepage": "https://github.com/pragkhard/namaste-reactjs#readme",
25   "devDependencies": {
26     "parcel": "^2.12.0",
27     "process": "^0.11.10"
28   },
29   "dependencies": {
30     "react": "^18.3.1",
31     "react-dom": "^18.3.1",
32     "react-router-dom": "^6.27.0"
33   },
34   "browserslist": [
35     "last 2 version"
36   ]
37 }

```

Routing configuration-

1. Import the createBrowserRouter and it will create the routing configuration for us, we are creating the routing configuration inside an appRouter.

What is the configuration – it means that some information that will define what will happen on the specific route/path.

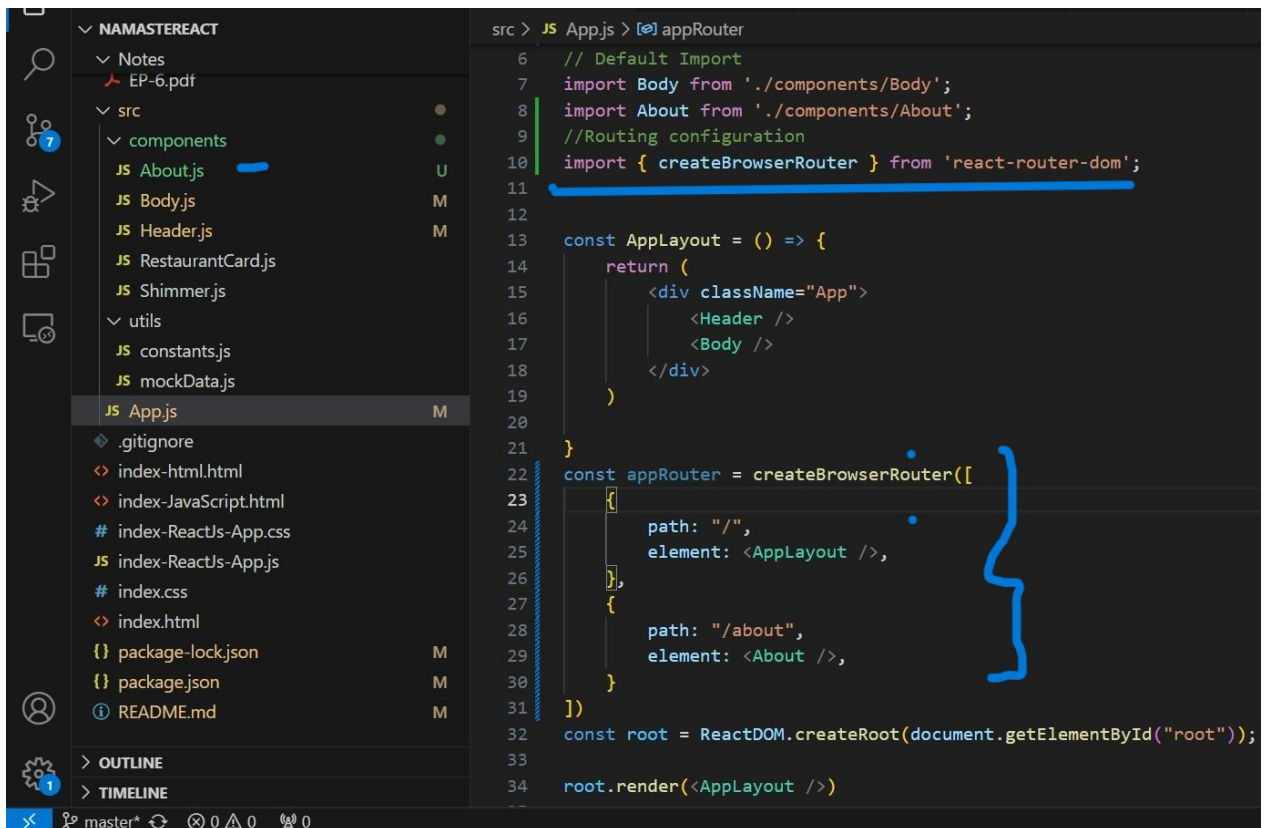
Suppose I will be calling about what should happen if I will call the url that we will pass inside the createBrowserRouter.

createBrowserRouter takes the list of paths and path is the object and it contains two things – path and element.

If my path is “/” then load the element/ load my Home page and here is my home page is AppLayout and if my path is “/about” load the about page/component.

Inshort –

1. Import createBrowserRouter from react router dom.
2. This createBrowserRouter takes some configuration, configuration is the list of objects and each and every objects define a different path and what should happen on the path.



```
src > JS App.js > [?] appRouter
6 // Default Import
7 import Body from './components/Body';
8 import About from './components/About';
9 //Routing configuration
10 import { createBrowserRouter } from 'react-router-dom';
11
12
13 const AppLayout = () => {
14   return (
15     <div className="App">
16       <Header />
17       <Body />
18     </div>
19   )
20 }
21
22 const appRouter = createBrowserRouter([
23   {
24     path: "/",
25     element: <AppLayout />,
26   },
27   {
28     path: "/about",
29     element: <About />,
30   }
31 ])
32 const root = ReactDOM.createRoot(document.getElementById("root"));
33
34 root.render(<AppLayout />)
```

When I create this configuration, I need to provide this configuration, why we need to provide we need to provide this for render it. How do I provide this for this we have one more important component that I import from react router dom.

React router dom is the functionality by given us the component which is known as routerProvider. This routerProvider will actually provide the routing configuration to our app “appRouter”.

How? – Earlier we were just render the <AppLayout /> directly instead of this we will provide the router configuration and provide the appRouter configuration
router={appRouter}

```

JS App.js > [?] appRouter
import Body from './components/Body';
import About from './components/About';
//Routing configuration
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

const AppLayout = () => {
  return (
    <div className="App">
      <Header />
      <Body />
    </div>
  )
}

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

const root = ReactDOM.createRoot(document.getElementById("root"));

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

```

1. Imported routerProvider this component exported from react-router-dom library. who wrote this code for us , this code would have been written by react-router-dom library people. The people who developed react-router-dom given us these features createBrowserRouter, routerProvider.
2. Now when we provided this feature to our app provider then it will be start working.

Just like createBrowserRouter there are multiple types of routers-

Routers

Picking a Router NEW

`createBrowserRouter` NEW

`createHashRouter` NEW

`createMemoryRouter` NEW

`createStaticHandler` NEW

`createStaticRouter` NEW

`RouterProvider` NEW

`StaticRouterProvider` NEW

createBrowserRouter

This is the recommended router for all React Router web projects. It uses the DOM History API to update the URL and manage the history stack.

Why we are using `createBrowserRouter`- because the documentation itself recommended it.

Create the boiler plate of the component - > rafce

We have created about and contact page.

What if we use random url , It will throws an unexpected error

The screenshot shows a web browser at `localhost:1234/pragati`. The page displays an "Unexpected Application Error!" with a "404 Not Found" status. Below the error message, it says "Hey developer" and provides a tip: "You can provide a way better UX than this when your app throws errors by providing your own `ErrorBoundary` or `errorElement` prop on your route."

The Chrome DevTools console shows the following error details:

- Warning: No routes matched location `"/pragati"` (history.ts:501)
- Error handled by React Router default ErrorBoundary: `ErrorResponseImpl {status: 404, statusText: 'Not Found', internal: true, data: 'Error: No route matches URL "/pragati"', error: Error: No route matches URL "/pragati" at getInternalRouterError (http://localhost:1234/index.7...)}` (index.js:1)

It is showing link this way, here it is showing 404 not found not showing the random ugly error on the screen but who is given the information all it will handle by react-router-dom. react-router-dom has created a error page for you. But this is the page react has created.

Let us create the own error page | customize error page-

We will add the errorElement:

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

If the path is this show the element, if there is an error load the error component.

Add hook feature

- There is one more feature , will use hook (useRouteError), This is the hook given by react router dom, using this hook it give us the more information about the error. So instead of the message we have written in the error component , it will tell the more detail, a better message an a page. So it will give the route fail error, network error it will give the multiple types of error.

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

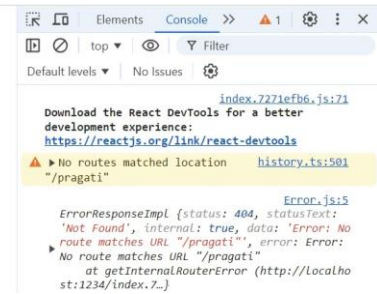
const Error = () => {
  const err = useRouteError();
  console.log(err)

  return (
    <div>
      <h1>OOps!!</h1>
      <h2>Something went wrong</h2>
    </div>
  )
}

export default Error;
```

OOps!!

Something went wrong



Will show the status on screen-

```
1 import { useRouteError } from "react-router-dom";
2
3 const Error = () => {
4   const err = useRouteError();
5   console.log(err)
6
7   return (
8     <div>
9       <h1>OOps!!</h1>
10      <h2>Something went wrong</h2>
11      <h3>{err.status}: {err.statusText}</h3>
12    </div>
13  )
14 }
15
16 export default Error;
```

Develop the children route-

What if I want to keep header intact and I just to my pages change below my header. Suppose I am on about page , so about page should load below the header. So for do this type of functionality will do children routes.

Will give the configuration children-

And the children again have list, list of paths

Previous-

```
JS App.js > [?] appRouter
const AppLayout = () => {
}

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    children: [],
    errorElement: <Error />
  },
  {
    path: "/about",
    element: <About />,
  },
  {
    path: "/contact",
    element: <Contact />,
  }
])

const root = ReactDOM.createRoot(document.getElementById("root"));
```

Now-

```

}
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    children: [
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      }
    ],
    errorElement: <Error />
  },
])

const root = ReactDOM.createRoot(document.getElementById("root"));

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

Creating children of AppLayout like about and contact are the children of AppLayout
After that render the children over there accordingly. Put some condition over there

```
const AppLayout = () => {
  return (
    <div className="App">
      <Header />
      { /* If path = / -> body should be there and about and contact should not be there */ }
      <Body />
      { /* If path = /about , I should have my about component over here and nothing */ }
      <About />
      { /* If path = /contact , I should have my contact component over here and nothing */ }
      <Contact />
    </div>
  )
}
```

So basically, what I want to push my children over here according to my route. If my children is /about I want this element to go app component, If my child component is /contact I want to get my contact element to come in this outlet. If I want my path / then I want body component to load in this outlet.

But how should I load all the children in this outlet, react router dom give us something Outlet, this Outlet is the component. How do we use, will create the outlet over there, where is the Outlet it is in the AppLayout so whenever there is change in the path so the outlet will be filled with the children according to the path what page you are.

So, whenever we use "/" my body will be filled in this outlet, whenever we use "/about" about will be filled in this outlet.

Outlet is like a tunnel all the children according to the routes go inside and come over here in this outlet.

About page went inside the outlet over here below the header, what I'm trying to do is my header is intact my outlet filled with the component that is pass as an child to my parent and the about page will load, We can have multiple parents, multiple children according to route it will render.

In this way we are creating the children route and this is how load the children routes inside the outlet.

Q. can we see the outlet in the HTML?

No because outlet is replaced by the component, so the component is showing in the outlet. So, the end user does not even get to know we have some outlet in the code.

In this way we are header is over there and we are reloading the pages accordingly to the routing. In this way we are using the children route in our app.

Linked to the pages –

Using anchor tag (Not recommended way)

```
<li>  
    <a href="/about"> About us</a>  
</li>
```

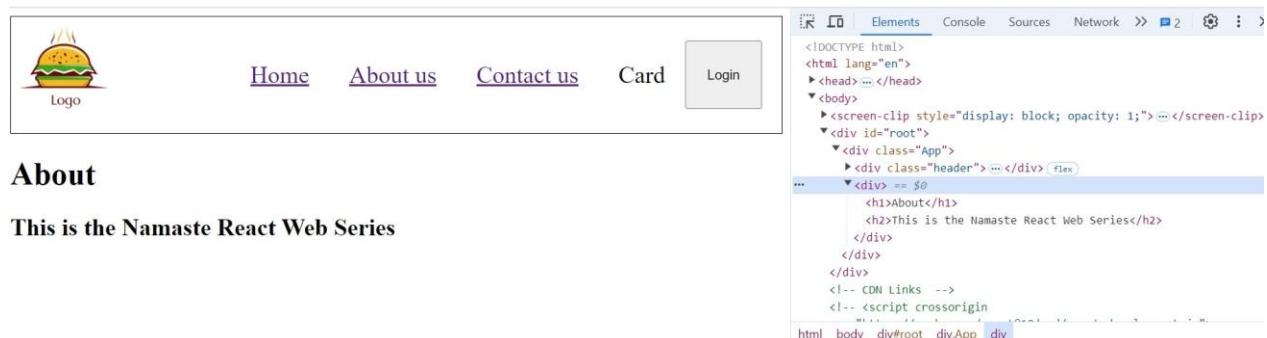
When we are using react and we want to route to some other page never use the anchor tag.

Why- suppose when I go to the aboutus page, the whole page got refresh but I don't want to refresh my whole page but still I want to move to the new page.

Note: - We can navigate to the new page without reload the whole page so instead of using anchor tag will use something call link component and this link comes from react-router-dom and link component works exactly like the anchor tag.

Difference-

The only difference between the anchor tag and link is attribute in anchor tag we are using "href" and in link we are using "to" and give the path where we want to navigate.



my whole page is not getting refreshed when I go to the about it will not reload the full page, It will fetches the data it does not reload the whole page it just changing the components. So, If I go to the next page my page will not be reloaded only this body will be replace by about component.

This is blinking because it is showing the full hierarchy where the changes happening. But it is not touching header. Header remains the same.

And when we use the anchor tag it will reload the whole page, If I use the link, it does not reload the whole page. It just refreshes the components; just changes the components.

That's why our react application know as single page application. It a whole single page/ component "AppLayout" and all the routing, all the new pages are just component interchange themselves.

Only one page if I go to the new route also it just changing the component its not reload the new page.

2 types routing in web apps-

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

Server-side routing means you have the index.html, about.html, contact.html, If I will click on my anchor tag it reload the whole page it sends the network call to about.html fetches the html and render to the webpage and refresh the whole page.

In short – make the network call and the page about.html coming from server.

Here **we are using the client-side routing**, there is no networks call when we are moving to the next page because all the component already loaded on to the app. There is only network call is when I made the API call known as client-side routing.

Create the dynamic route for all the restaurant we have-

```
{  
    path: "/restaurants/:resId",  
    element: <RestaurantMenu />,  
}
```

:resId is dynamic it can be changed according to the id of restaurant and every restaurant has different id, so our routes are unique whenever we have new restaurant.

Like this way we can see on browser-

<http://localhost:1234/restaurants/123>

Created the restaurant menu component-

Make the API call and update the variable and add the Shimmer effect.

```

const [resInfo, setResInfo] = useState(null);

useEffect(() => {
  fetchMenu();
}, [])

const fetchMenu = async () => {
  const data = await fetch("https://www.swiggy.com/dapi/menu/pl?page-
type=REGULAR_MENU&complete-
menu=true&lat=18.6161&lng=73.7286&restaurantId=14780&catalog_qa=undefined&s
ubmitAction=ENTER")

  const json = await data.json()

  console.log(json);
}
// if (resInfo == null) return <Shimmer />
OR
return (resInfo == null) ? <Shimmer /> : (

```

resInfo is null initially as soon as my data call and I have got the json , I will fill this resInfo with json.data.

I will show the data of the restaurant-

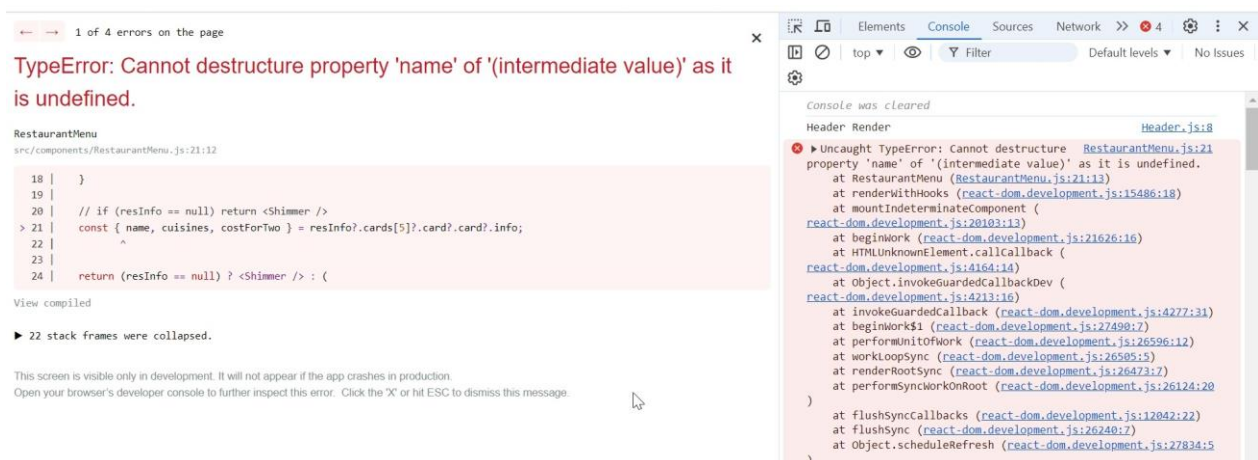
But when I Destructuring it I am getting error –

```
const RestaurantMenu = () => {
  const fetchMenu = async () => {

  }

  // if (resInfo == null) return <Shimmer />
  const { name, cuisines, costForTwo } = resInfo?.cards[5]?.card?.card?.info;

  return (resInfo == null) ? <Shimmer /> : (
    <div className="menu">
      <h1>{name}</h1>
      <p>{cuisines?.join(", ")}- {costForTwo}</p>
      <h2>Menu</h2>
      <ul>
        <li>Biryani</li>
        <li>Burgers</li>
        <li>Diet Cake</li>
      </ul>
    </div>
  )
}
```



resInfo is null and my code is trying to find the name inside my null that's why getting this error. So first we return shimmer and remove the tertiary condition shimmer.

So, when my code will be executed it won't even go after this line if (resInfo == null) return <Shimmer />, it will return shimmer, and we will not get this error. No use of ternary operator here.

Don't use ternary operator blindly.

Another point –

Every restaurant has different apis

Example- this api is for pizza hut

https://www.swiggy.com/dapi/menu/pl?page-type=REGULAR_MENU&complete-menu=true&lat=18.6161&lng=73.7286&restaurantId=14780&catalog_qa=undefined&submitAction=ENTER

this api is for burger king

https://www.swiggy.com/dapi/menu/pl?page-type=REGULAR_MENU&complete-menu=true&lat=18.6161&lng=73.7286&restaurantId=253596&catalog_qa=undefined&submitAction=ENTER

“The only difference between it is restaurantId”

We can change the api code and see the differences.

Now I want to pass this id from my routes. Suppose if I use /14780 in the url it will load pizza hut and if I use /253596 it will load burger king.

But I do I read the id in my component?

Again, we have superpower given to us by react-router-dom which will give us what is there as a param, the superpower is the hook which is known as “useParam”

Here I am using /123, check in the console will show /123 and after that use that id in the URL.

So, according to the url my data has changed

Steps-

1. import { useParams } from "react-router-dom";
2. In App.js file where I have gave the route path, same resId I have to use in useParams hook

```
{
  path: "/restaurants/:resId",
  element: <RestaurantMenu />,
}
```

3. // destructuring

```
const { resId } = useParams()
console.log(resId)
```

4. Pass the resId after the Api url like –

```
const data = await fetch(MENU_API + resId)
```

MENU_API = https://www.swiggy.com/dapi/menu/pl?page-type=REGULAR_MENU&complete-menu=true&lat=18.6161&lng=73.7286&restaurantId=

Now make the restaurant clickable –

For this will make the Link component from react-router-dom and will link the restaurant cards and in link give the route path.

So we have built dynamic routes, dynamic url for every restaurant we have and we have link it using a link. We are using link instead of anchor tag. But when we inspect it showing in link we will find the anchor tag instead of link in DOM.



Link is the component which is given us to the react-router-dom which is the special component and behind the scenes link is using anchor tag. So link is basically the wrapper of the anchor tag. When you make a link that means now react-router-dom will keep the tag that “Home” is the link and you don’t refresh the page.