

9 . Optimising Our App

Creating own custom Hook

Why we need to create custom Hook?

Creating custom Hook is not mandatory thing but it's good thing because that will make our code more readable , modular , reusable

- 1.fetching the data
- 2.displaying the data

Problem

Getting this error



Solution

We writing this code in better way using custom Hooks

Single responsibility principles

Suppose if we have function , class or any single identity that should have single responsibility

So here each component has different function so according to that principle each of this component should have single responsibility

Suppose

Eg if component

RestarantMenu

It should only have responsibility to displaying restaurant Menus

RestarantCard

It should only have responsibility to displaying restaurant cards

Modularity

We breakdown our code into different different small small modules so that our code more maintainable and more testable

So if we follow single responsibility principles then we got following features

1. More Reusability
2. More Maintainable
3. More testable

Hooks are just Utility functions

Hook made more more readable , more modular and more reusable
Hooks are helper function

So best way to create helper function in utils

Creating custom Hook

We creating custom hook for fetching data (Following code)

```

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

const fetchMenu= async()=>{
const data= await fetch(MENU_API +resId
);
    const json = await data.json();
    console.log(json);
    setResInfo(json.data)

};

```

So removing this Logic and instead of creating custom Hook

That will fetch the data and will give it to restaurantMenu (Component)
So we Passing our resId so it will fetch the data

Create file useRestarantMenu (custom hook) in utils

```

JS RestaurantMenu.js U  JS useRestarantMenu.js U X
9. Optimising Our App > src > utils > JS useRestarantMenu.js > useRestarantMenu
1  const useRestarantMenu = (resId)=>
2  {
3
4
5      return resInfo;
6  }
7
8

```

Here we get resId we want resInfo

```
JS RestarantMenu.js U JS useRestarantMenu.js U X Settings JS Body.js U
9. Optimising Our App > src > utils > JS useRestarantMenu.js > useRestarantMenu > fetchData
1 import { MENU_API } from "../constants";
2 import { MENU_API } from "../utils/constants";
3 import { useState,useEffect } from "react";
4
5 const useRestarantMenu = (resId)=>
6 {
7   const [resInfo,setResInfo]=useState(null);
8
9
10   //fetchdata
11   useEffect(()=> {
12     fetchData();
13   },[]);
14
15   const fetchData = async()=> {
16     const data= await fetch(MENU_API+resId);
17     const json = await data.json();
18
19     setResInfo(json.data);
20   };
21
22   return resInfo;
23 };
24
25 export default useRestarantMenu
```

```
JS RestarantMenu.js U X JS useRestarantMenu.js U JS Body.js U
9. Optimising Our App > src > Components > JS RestarantMenu.js > ...
1 import Shimmer from "../Shimmer";
2 import {useParams} from "react-router-dom";
3 import useRestarantMenu from "../utils/useRestarantMenu";
4
5
6
7 const RestarantMenu=()=>
8 {
9
10   const {resId}=useParams();
11   const resInfo=useRestarantMenu(resId);
12
13
14
15
```



Getting this error so to resolve that use &&

So the problem you are facing is you are trying to map over the data before it becomes available (as per your logs) what you want to do is make sure the data is available before blindly mapping over it... You can do such by checking that the data is available. You can learn more about conditional rendering [here](#).

Just add a && (AND) clause to your statement, so that you can guarantee that data is not undefined and available.

This will give you the ability to wait for the data to load from your server.

So the statement becomes `data && data.map(...)`

So your code will look like the following.

```
{data && data.map((forecast, index) => (
  <div key={index}>

    <p>{forecast[0]}</p>

  </div>
)}
```

Building online offline feature

Custom hook Note

When we creating own custom hook need to two things remember

1. What is the **input**
2. what will be **output**

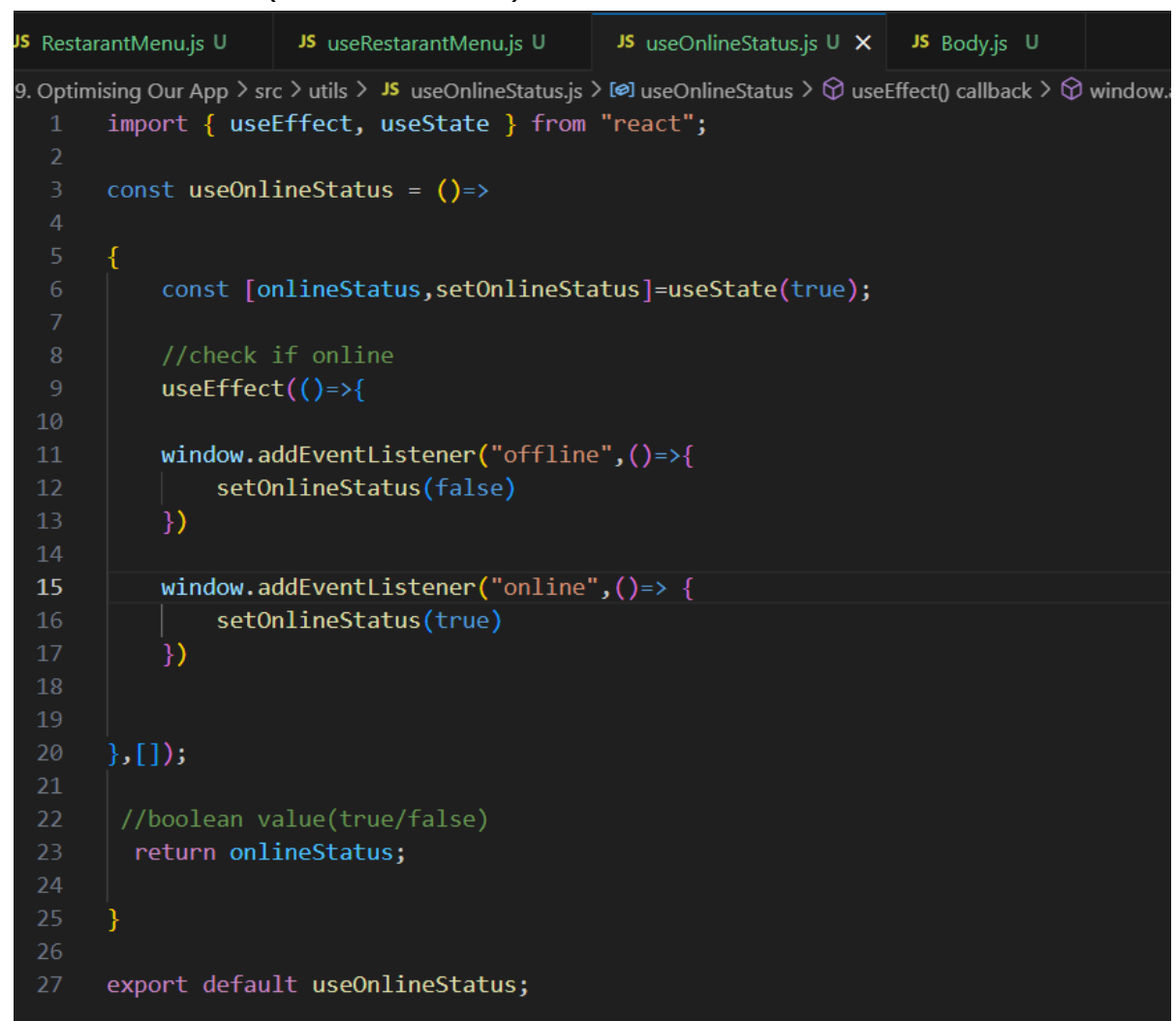
Here we don't need of any input

For check user online or offline we use **online event listener**

We need to only one time to add online event listener

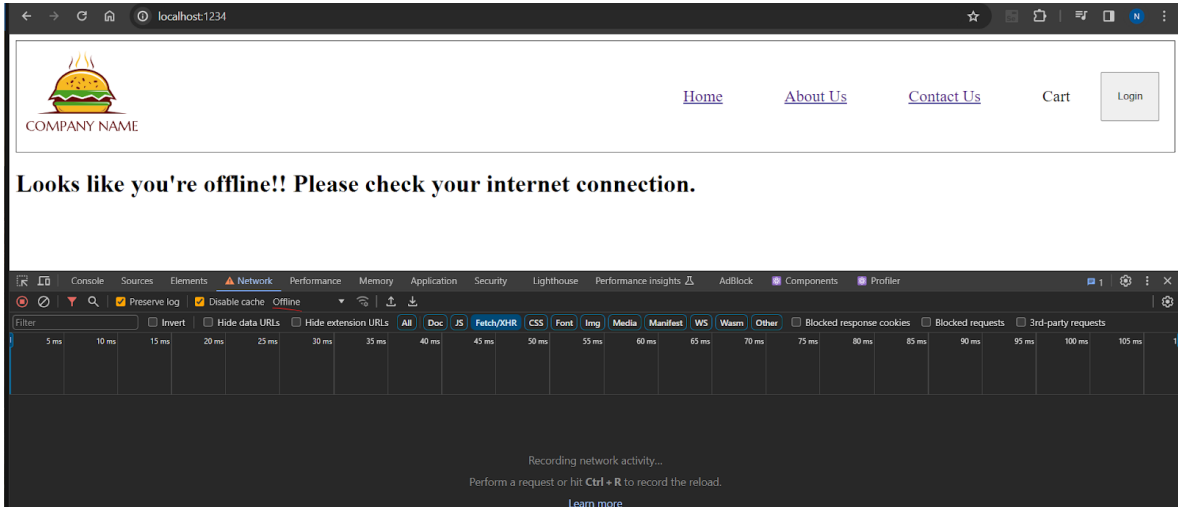
So we use **useEffect** hook

Our custom hook (useOnlineStatus)



```
JS RestarantMenu.js U JS useRestarantMenu.js U JS useOnlineStatus.js U X JS Body.js U
9. Optimising Our App > src > utils > JS useOnlineStatus.js > useOnlineStatus > useEffect() callback > window.
1  import { useEffect, useState } from "react";
2
3  const useOnlineStatus = ()=>
4
5  {
6      const [onlineStatus,setOnlineStatus]=useState(true);
7
8      //check if online
9      useEffect(()=>{
10
11          window.addEventListener("offline",()=>{
12              setOnlineStatus(false)
13          })
14
15          window.addEventListener("online",()=> {
16              setOnlineStatus(true)
17          })
18
19
20      },[]);
21
22      //boolean value(true/false)
23      return onlineStatus;
24
25  }
26
27  export default useOnlineStatus;
```

So when online it's look normal but when offline we got that msg



Or

Online green dot
Offline red dot

```
JS RestarantMenu.js U JS useRestarantMenu.js U JS useOnlineStatus.js U X JS Body.js U JS Head...
9. Optimising Our App > src > utils > JS useOnlineStatus.js > useOnlineStatus > useEffect() callback > window.a
1  import { useEffect, useState } from "react";
2
3  const useOnlineStatus = ()=>
4
5  {
6      const [onlineStatus,setOnlineStatus]=useState(true);
7
8      //check if online
9      useEffect(()=>{
10
11          window.addEventListener("offline",()=>{
12              setOnlineStatus(false)
13          })
14
15          window.addEventListener("online",()=> {
16              setOnlineStatus(true)
17          })
18
19      },[]);
20
21      //boolean value(true/false)
22      return onlineStatus;
23
24  }
25
26
27  export default useOnlineStatus;
```

```
9. Optimising Our App > src > Components > JS Header.js > Header
1  import { LOGO_URL } from "../utils/constants";
2  import { useState } from "react";
3  import { Link } from "react-router-dom";
4  import useOnlineStatus from "../utils/useOnlineStatus";
5
6  const Header =() =>
7  {
8
9      const [btnName, setbtnName]=useState("Login ")
10
11      const onlineStatus= useOnlineStatus();
12
13
14      return (
15
16          <div className="header">
17              <div className="logo-container">
18                  <img className="logo"
19                      src={LOGO_URL} alt="" />
20              </div>
21
22              <div className="nav-items">
23                  <ul>
24
25                      <li>
26                          online Status : {onlineStatus?"🟢":"🔴"}
27                      </li>
28
29                      <li>
30
```


When we build large scale application the performance was not good

Problem with large scale app
Dist make 1 file of all files

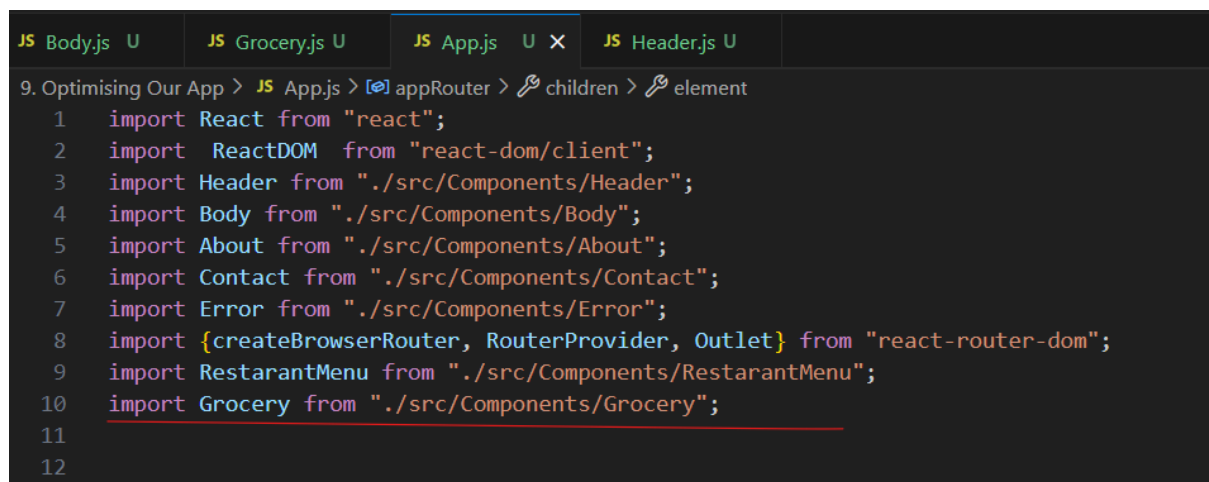
Our js files increases by how many component it holds

Should be bundelling or not , yes of course because if we 1000 files we don't want 1000 of component loading in other pages and we don't want 1000 files In 1 file both of solution not true so

We will try to make smaller bundle of this files this process is known as
Chunking || Code Splitting || Dynamic Bundelling || lazy Loading || On demand Loading || Dynamic import

Creating Grocery component

Suppose Grocery business has lot of child component
So we want logically distribute our application



```
JS Body.js U JS Grocery.js U JS App.js U X JS Header.js U
9. Optimising Our App > JS App.js > [e] appRouter > [e] children > [e] element
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import Header from "./src/Components/Header";
4  import Body from "./src/Components/Body";
5  import About from "./src/Components/About";
6  import Contact from "./src/Components/Contact";
7  import Error from "./src/Components/Error";
8  import {createBrowserRouter, RouterProvider, Outlet} from "react-router-dom";
9  import RestarantMenu from "./src/Components/RestarantMenu";
10 import Grocery from "./src/Components/Grocery";
11
12
```

So we don't import Grocery directly like this

But we importing Grocery using lazy Loading

So initially our code was not grocery code when we go to the Grocery page/Link then only Grocery code come up

So that is known as lazy loading

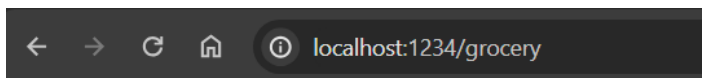
We will not loading everything directly but we do a lazy loading when required and that is also known as on demand loading

We importing Grocery using lazy loading like this

```
9. Optimising Our App > JS App.js > AppLayout
1  import React,{lazy} from "react";
2  import ReactDOM from "react-dom/client";
3  import Header from "./src/Components/Header";
4  import Body from "./src/Components/Body";
5  import About from "./src/Components/About";
6  import Contact from "./src/Components/Contact";
7  import Error from "./src/Components/Error";
8  import {createBrowserRouter, RouterProvider, Outlet} from "react-router-dom";
9  import RestarantMenu from "./src/Components/RestarantMenu";
10 // import Grocery from "./src/Components/Grocery";
11
12
13
14 const Grocery = lazy(()=>import("./src/Components/Grocery"));
15
```

Lazy is named import

We loading Grocery code on demand when we click on that then only our Grocery code will be loading
Then react throwing an error



Opps!!!

Something Went Wrong!!

:

Why

The grocery code take 12 Millisec to come to the browser (fetching the data of grocery) but react is very fast , react try to load Grocery component but code was not there so throwing this error

To handle this type of error we will use **Suspense** (to handle that state)

```
import React,{lazy, Suspense} from "react";
import ReactDOM from "react-dom/client";
import Header from "./src/Components/Header";
import Body from "./src/Components/Body";
import About from "./src/Components/About";
import Contact from "./src/Components/Contact";
import Error from "./src/Components/Error";
import {createBrowserRouter, RouterProvider, Outlet} from "react-router-dom";
import RestarantMenu from "./src/Components/RestarantMenu";
// import Grocery from "./src/Components/Grocery";
```

```
{
  path: "/grocery",
  element: <Suspense fallback={<h1>Loading...</h1>}><Grocery/></Suspense>>,
},
```

We have to give **Suspense** fallback to what should react render when the code not available

Eg . we are on home page so code of Grocery not there , when we clicking on Grocery so react try to load something it can not load until the Grocery code is there so meanwhile intermediately react want to something present on the screen & we can give that inside the fallback of Suspense

So inside **fallback** we need to pass jsx , so we can pass Shimmer UI also

