**Single responsibility principle**

What is it?

Suppose if we have function that should have single responsibility

1.Each component should have separate responsible

2.all the functionality shouldn’t be in single component

3.modular fashion -> code is split into multiple small component and it Is easy to manage

4.when we break the code into small pieces then it is easy to write test cases and easy to check for bugs

5.if we use the modular concept the code becomes

1.resuable

2.maintainable

3.testable

6.There are no rules for single responsibility principle 🡪 we should write code so that any external user should be able to understand the purpose of the component and understand easy

What is custom hooks and its importance?

* Hooks are like utility functions with extra powers
* Custom hooks are just functions with some responsibility that helps to bring modularity to the code

 **DRY (Don't Repeat Yourself)** — reuse logic instead of copying it

 Clean up components — move logic into separate files/functions

 Share stateful logic across multiple components

* Hooks are helper functions and always prefer to create sepreate file for separate hooks
* Always start with use as start name

// custom hooks are normal js functions

// the logic of geting the data in restramenu is extracted to hook and no need to worry about it

import { useState, useEffect } from "react";

const useRestaurantMenu = (resId) => {

  //add logic to get restro Information and resturn it

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

  useEffect(() => {

    fetchRestroData();

  }, []);

  const fetchRestroData = async () => {

    try {

      const data = await fetch(

        `https://www.swiggy.com/dapi/menu/pl?page-type=REGULAR\_MENU&complete-menu=true&lat=11.0168445&lng=76.9558321&restaurantId=${resId}&catalog\_qa=undefined&submitAction=ENTER`

      );

      const response = await data.json();

      setResInfo(response.data);

    } catch (err) {

      console.log(err);

    }

  };

  return resInfo;

};

export default useRestaurantMenu;

**we can create a custom hook by ourself and publish it -🡪 make a try with concepts**

**optimizing the app**

1.when we use bundler like parcel it will make all the files into single file as bundle in dist folder

2.the problem of having single file for all the components is the size of single js file will be very huge if the application is very huge .. to overcome this we need to make some alternate to make it look good

3.we need to break app into small peace’s and make few files instead of single file

We will try to create small bundles and we can do it using chunking and this chunking has many names as below

* Code splitting
* Lazy loading
* Chunking

How to do chunking and when to do it?

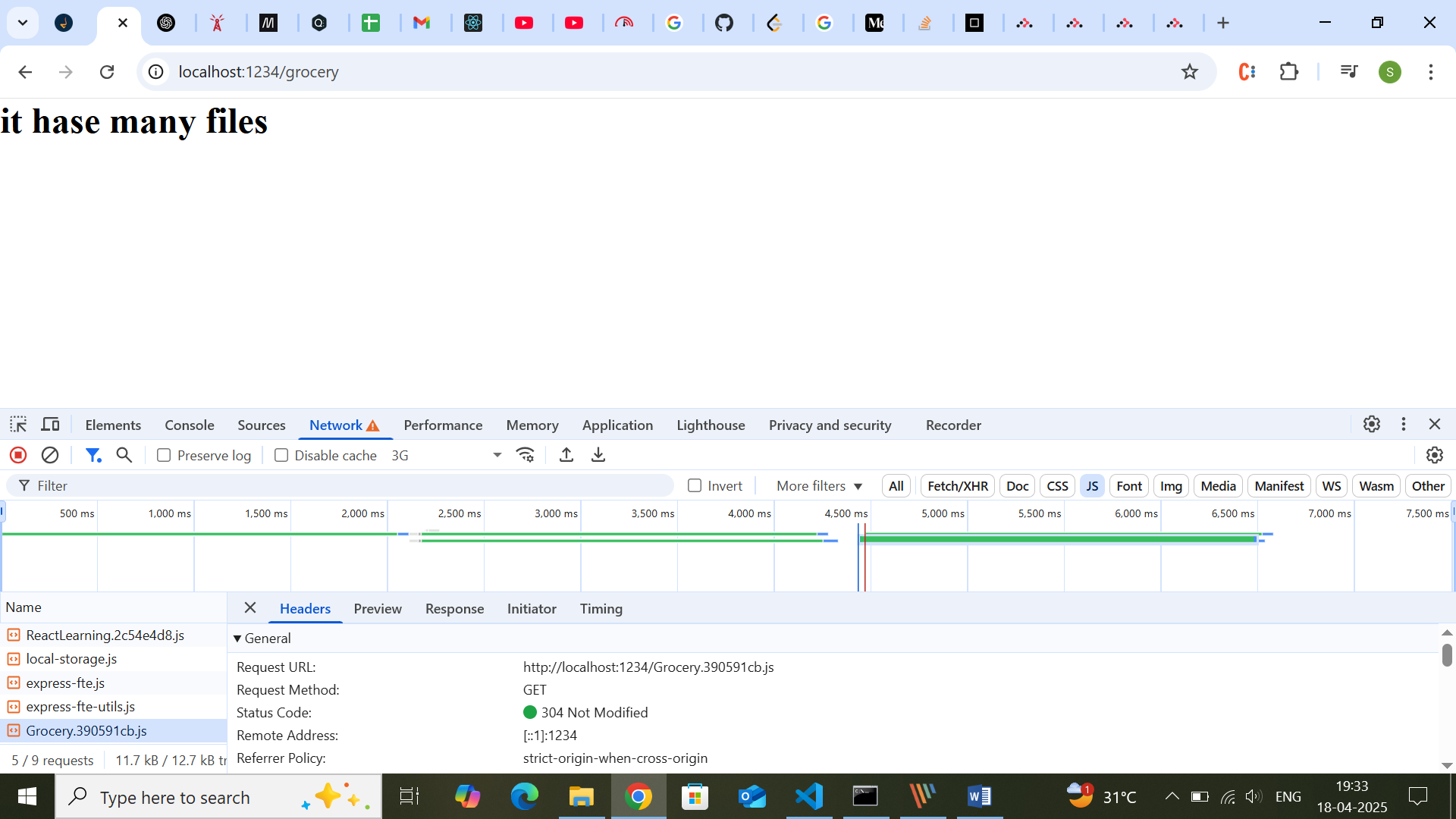
* The bundle must have some purpose and should have some logic eg: for makemy trip
  + Flight
  + Book cab
  + Book vechicles

In react we have a function called **lazy()** which will not load the component on first time when the app is loaded but it loads the component only we try to access it using navigation or load page

// when the component is loaded and when it is taking time to load it iwll throw error in ui to solve it we use suspends

const Grocery = lazy(() => import("./components/Grocery"));

This above code creates a sepreate file with grocery component alone when it is loaded



Meanwhile when the grocery is loaded the react will throw error as there is no ui rendered at meantime the grocery is loaded so to avoid it there is component provided by react it is called **suspense**

<Suspense fallback={<div>The grocery is loading</div>}>

<Grocery/>

<Supsense/>

The component takes props as fallback jsx which is loaded meantime till grocery is loaded

 {

    path: "/grocery",

    element: (

      <Suspense fallback={<Shimmer />}>

        <Grocery />

      </Suspense>

    ),

  },

Here shimmer is component that shows the loading error message

| **Use Case** | **Lazy Load? ✅** |
| --- | --- |
| Page-level routes (via React Router) | ✅ Yes |
| Rarely used modals/popups | ✅ Yes |
| Heavy dashboard/reporting components | ✅ Yes |
| Admin/settings pages | ✅ Yes |
| Header/Footer/Navbar | ❌ No |
| Main home components | ❌ No |
| Frequently used components | ❌ No |

1. **Use lazy loading for route-based code splitting**  
   → e.g., use it in React Router to load pages on demand.
2. **Don’t lazy load critical UI**  
   → Like headers, footers, navigation menus — they should be instantly available.
3. **Keep chunk sizes reasonable**  
   → Don't split too much — it can backfire.

| **Lazy Loading Rule** | **Recommendation** |
| --- | --- |
| Use for large, infrequent components | ✅ Yes |
| Use for routes/pages via React Router | ✅ Yes |
| Use for modal dialogs, charts, or admin panels | ✅ Yes |
| Use for frequently used UI like buttons/navbar | ❌ No |
| Use for tiny components | ❌ Overkill |