# Chapter 09: Optimising our App

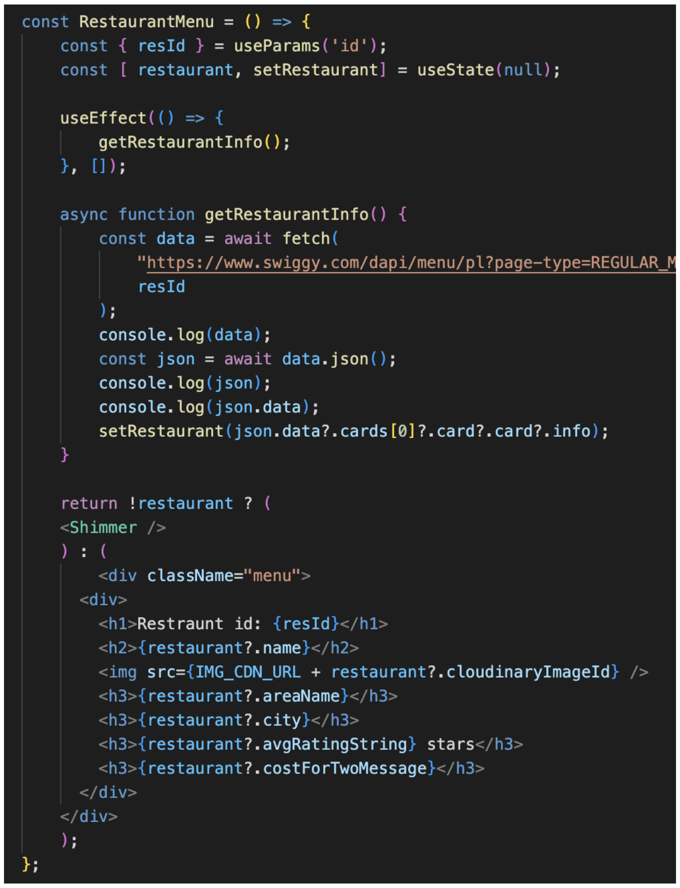
## ▼ Single responsibility principle

- Eg: RestaurantCard should be showing only restaueant info - single responsibility
- If we are writing multiple things in single component, we need to break it up in to different components
- Makes it modular. This makes it to easy to unit test and regression test too.
- Easy to maintain too
- Also resuability - smaller components with single functionality can be easily reused
- No hard and fast rule for Single responsibility principle

## ▼ Custom Hooks

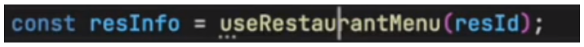
- Hooks are special Javascript functions React gave us (useState(), useEffect() etc)
- We can also create our own custom hooks too
  - to make compoents adhere to Single responsibility principle
  - extract extra responsibility from these components, and extract it to inside custom hook
  - Custom Hooks are like utility/helper functions

    Eg:
    RestaurantMenu  Component:



```
const RestaurantMenu = () => {
    const { resId } = useParams('id');
    const [ restaurant, setRestaurant] = useState(null);

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

    async function getRestaurantInfo() {
        const data = await fetch(
            "https://www.swiggy.com/dapi/menu/pl?page-type=REGULAR_M
            resId
        );
        console.log(data);
        const json = await data.json();
        console.log(json);
        console.log(json.data);
        setRestaurant(json.data?.cards[0]?.card?.card?.info);
    }

    return !restaurant ? (
    <Shimmer />
    ) : (
        <div className="menu">
      <div>
        <h1>Restraunt id: {resId}</h1>
        <h2>{restaurant?.name}</h2>
        <img src={IMG_CDN_URL + restaurant?.cloudinaryImageId} />
        <h3>{restaurant?.areaName}</h3>
        <h3>{restaurant?.city}</h3>
        <h3>{restaurant?.avgRatingString} stars</h3>
        <h3>{restaurant?.costForTwoMessage}</h3>
      </div>
    </div>
    );
};
```

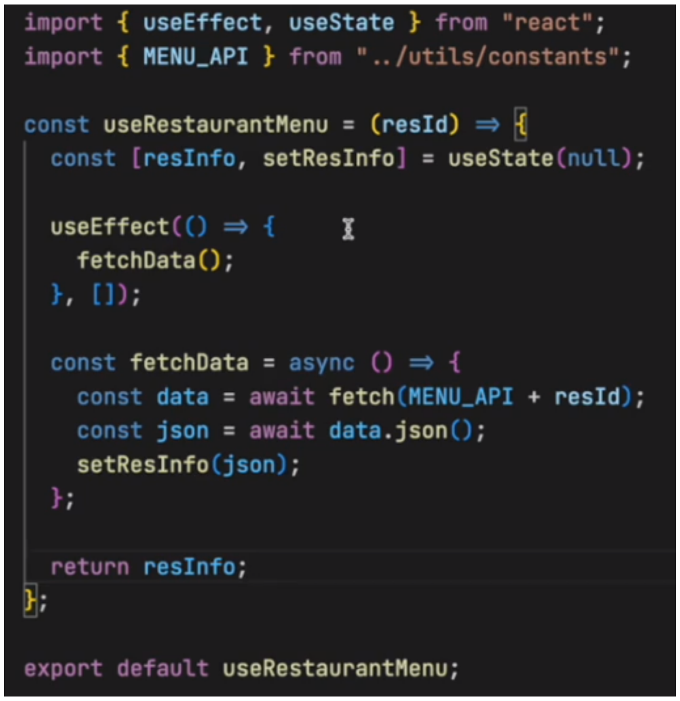
- Two responsibility: Fetches data, & display data
- Extract Fecthing data logic in to a custom hook, so that component need not worry about getting data, managing state. It can just concentrate on displaying data
    - Some thing like



```
const resInfo = useRestaurantMenu(resId);
```

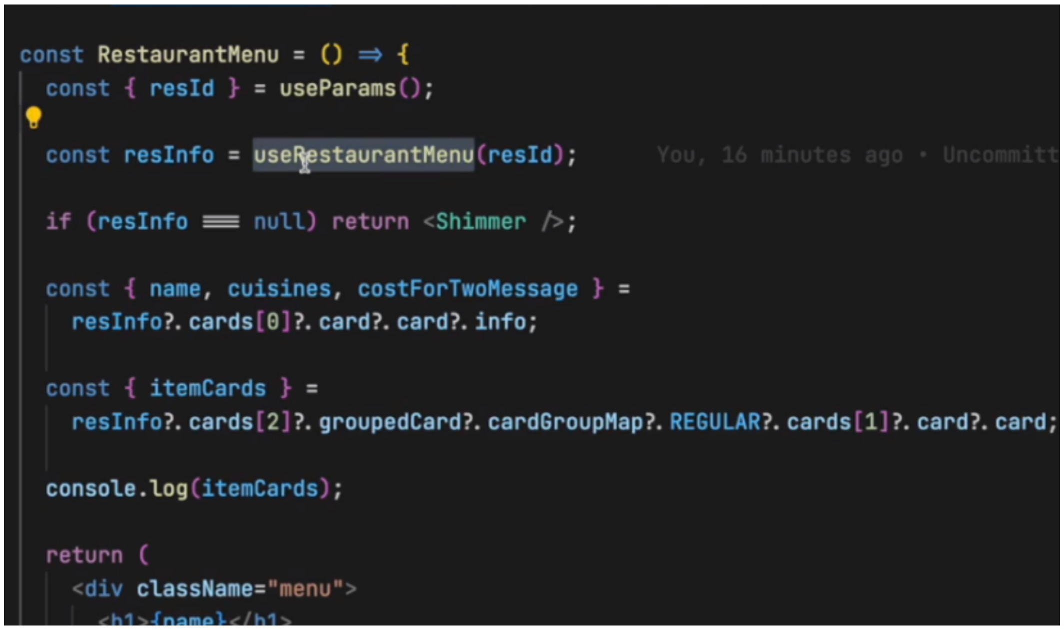
- Good practice while creating custom hook:
    - create it in utilities folder
    - Have separate file for each custom hoom
    - start the name with ‘use’.
        - It will work even its name differently not starting with ‘use’. Its just recommended way. Reader will now its hook, not normal function.

Also many linters are configured to through error if not



```javascript
import { useEffect, useState } from "react";
import { MENU_API } from "../utils/constants";

const useRestaurantMenu = (resId) => {
  const [resInfo, setResInfo] = useState(null);

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

  const fetchData = async () => {
    const data = await fetch(MENU_API + resId);
    const json = await data.json();
    setResInfo(json);
  };

  return resInfo;
};

export default useRestaurantMenu;
```

- This is same functionality as before. Just that code is extracted
- Also now we dont need to manage state in RestaurantMenu Component
- Now its easy to test. If there is any issue fetching data test useRestaurantMenu , test displaying data using RestaurantMenu Component

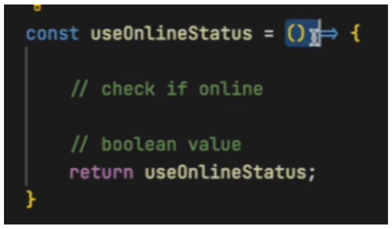


```
const RestaurantMenu = () => {
  const { resId } = useParams();

  const resInfo = useRestaurantMenu(resId);        You, 16 minutes ago • Uncommitt

  if (resInfo === null) return <Shimmer />;

  const { name, cuisines, costForTwoMessage } =
    resInfo?.cards[0]?.card?.card?.info;

  const { itemCards } =
    resInfo?.cards[2]?.groupedCard?.cardGroupMap?.REGULAR?.cards[1]?.card?.card;

  console.log(itemCards);

  return (
    <div className="menu">
      <h1>{name}</h1>
```

- My thoughts:
  - What is the differnce between normal functions and custom hooks we create?
    - custom hooks use react hooks (useState(), useEffect())
  - Since these dont return JSX in these function, what happens when state changes, and when does useEffect will run?
    - It think the state & useEffect will be added to the component that uses this custom hook. So its same as adding state, useEffect() directly to the component, just that logic is extracted out for easy readability, testability and reusability.
    - So when state changes the Component that is using this custom hook will rerender. And useEffect() from this custom hook will run after component is rendered

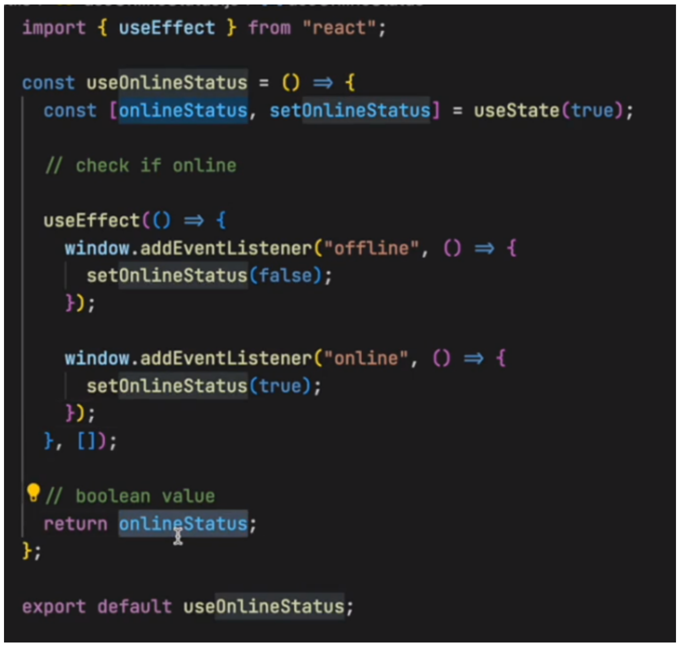
## ▼ Create a custom hook, to know whether the use is online or offine (connected to internet or not)

- To write custom hook, first finalise the contract (input/ output)



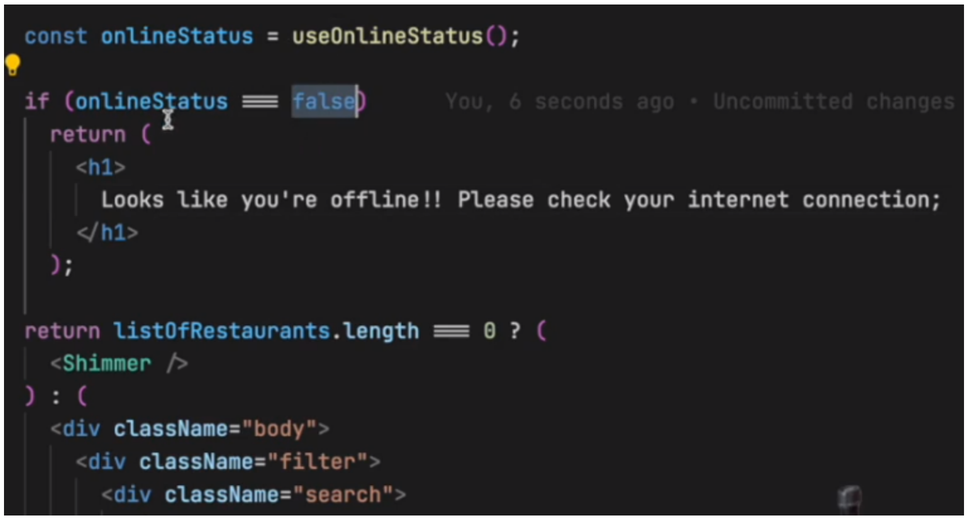
```
const useOnlineStatus = () => {

    // check if online

    // boolean value
    return useOnlineStatus;
}
```

- There is existing online/ offline event that gets triggered based on internet connectivity status. We need to add the listener for that, only once to your page. Add in useEffect



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

const useOnlineStatus = () => {
  const [onlineStatus, setOnlineStatus] = useState(true);

  // check if online

  useEffect(() => {
    window.addEventListener("offline", () => {
      setOnlineStatus(false);
    });

    window.addEventListener("online", () => {
      setOnlineStatus(true);
    });
  }, []);

  // boolean value
  return onlineStatus;
};

export default useOnlineStatus;
```

- Consume custom hook

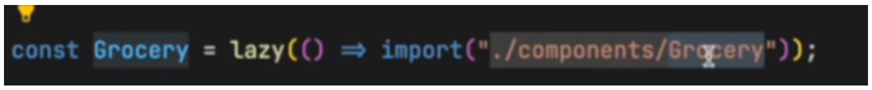


```
const onlineStatus = useOnlineStatus();

if (onlineStatus === false)        You, 6 seconds ago • Uncommitted changes
  return (
    <h1>
      Looks like you're offline!! Please check your internet connection;
    </h1>
  );

return listOfRestaurants.length === 0 ? (
  <Shimmer />
) : (
  <div className="body">
    <div className="filter">
      <div className="search">
```

## ▼ Code splitting / Lazy loading / Dynamic-bundling / Chunking / On demand loading / Dynamic import

- Parcel is used to bundle our app (along with other functionalities Iit does)
  - It bundles to one js file (along with map file that parcel uses in backend)
  - If app is big, size of this js file becomes large
- How to optimise this?
  - We can make small bundles
  - This is known in many name
    - chunking
    - code-spitting
    - dynamic-bundling
    - Lazy-loading
    - on demand loading
    - dynamic import
  - A bundle should have enough code for a functionality
- How to do this?
  - `import { lazy }` from `'React'`

- lazy takes a callback function

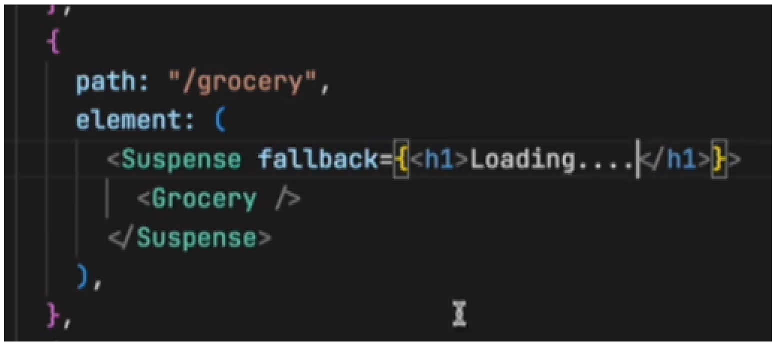


```
const Grocery = lazy(() => import("./components/Grocery"));
```

- `import` here is not exactly the one as we use at top. Its a function that takes path
- This just a one line code, but does lot of thinks
- When we hit grocery tab, main bundle will realize grocery code is not loaded and load that grocery js bundle

Error: A component suspended while responding to synchronous input

- When we click grocery tab, it will take some time to download bundle. But React tries to render during this time and throws error
- To overcome this- we use **`Suspense`** which is also imported from `'React'`
- We need to wrap Grocery component with `Suspense` and use `fallback` (react will render this during loading)



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

- This simple four lines code is very powerful, can make our large app performant, very fast, optimised
- This will be asked in interview in front end system design for senior engineer