#NOTES

- Read this to avoid common mistakes while using useState Link
- Link1, Link2 for CSS Flexbox (not extremely necessary, but a good read)
- Read about different types of dependencies from Link and also from Link
- Convert HTML to JSX using this tool :- Link
- Why browsersList is reqd in package.json Link
- React is faster because of Reconciliation algorithm.
- Whenever we use a state variable in a component, everytime React detects a change in the state variable, it rerenders the whole component, instead of just the part of the component which has a state variable, like :-

```
12  const Body = () => {
13      let [searchText, setSearchText] = useState("");
14      const [restaurants, setRestaurants] = useState(restaurantList);
15      console.log("Re-rendered");
16      return (
17          <>
18              <div className="search-container">
19                  <input
20                      type="text"
21                      className="search-input"
22                      placeholder="Search"
23                      value={searchText}
24                      onChange={(e) => {
25                          setSearchText(e.target.value);
26                      }}
27                  >
28                  </input>
29                  <button className="search-btn" onClick={() => {
30                      // need to filter the data
31                      const data = filterData(searchText, restaurants);
32                      // update the state
33                      setRestaurants(data);
34                  }}>Search</button>
35              </div>
36              {console.log("Re-rendered again")}
37              <div className="restaurantList">
```

```
const AppLayout = () => (
    <>
        {console.log("Rendered in AppLayout too")}
        <Header />
        <Body />
        <Footer />
    </>
)
```

In line 15 and 36 of Body component and in AppLayout component, we are console logging everytime we render the component. So, when the server is started for the first time, those 3 will get printed. Then, if you click on the Search button once, the whole Body component will get re-rendered , but the AppLayout will not because that's not the component with state variable. So, we will get only 2 console logs on clicking Search button once.

```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Rendered in AppLayout too
Re-rendered
Re-rendered again
Re-rendered
Re-rendered again
>
```

Exploring the world through APIs :-

In JS, we used to connect to APIs using the fetch() browserAPI/function provided to us by the windows object. Now, where to call the API? If you call the API inside a component with a state variable, every time the state variable changes, React will rerender that component and we will be making lots of unnecessary API calls. (for ex :- If I call an API inside the Body component of my project day_5, everytime I click the Search button, that API will be called).

So, we have to build a feature, so that every time our page loads, the API call is made and the data from it is filled into the required variable. Now there are still 2 ways to do this :-

i. When user loads the page, the API call is made (suppose it takes 300 ms to fetch data from that API) and then the components are rendered (suppose it takes another 200 ms to render the components with the API data). So, when user loads our webpage, he gets to see our UI, after 500 ms.

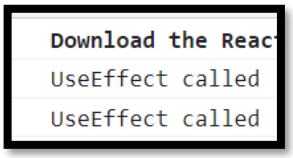
ii. When user loads the page, an initial component is rendered immediately on UI (suppose it takes 100 ms), then the API call is made (300 ms) and then with the API data, the UI is updated (takes another 200 ms). So, in total it takes 600 ms for the user to get the actual representation of out website, but the user experience is better here because he can see the initial components almost immediately instead of glaring a t a blank screen as in the 1st case.

We prefer the 2nd approach and to accomplish this, React gives us another functionality, or another hook called useEffect.

useEffect() Hook

This hook is also provided by the React library and injected by a named import. It has a callback function as it's 1st parameter, which is called everytime useState() is called and useState(), when defined within a component, is called everytime our component is rendered or re-rendered. We know that our component is re-rendered or rendered (I will be using the word rendered to specify both of these actions because I am lazy) whenever, we load a page or a state variable changes or we change the props. But, this is a bad way to call it.

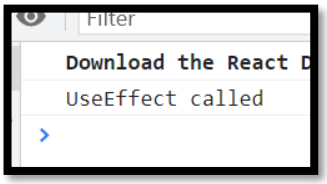
```
import { useState, useEffect } from "react";

function filterData(searchText, restaurants) {
    const filterData = restaurantList.filter(
        (restaurant) => restaurant.info.name.toLowerCase().includes(searchText.toLowerCase())
    );
    return filterData;
}

const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState(restaurantList);
    useEffect(() => { console.log("UseEffect called") })
    return (
        <>···
        </>
    )
};
```



The first output is when the page loads, the second is when we click the Search button once

To overcome this, we pass another parameter to the useEffect function, called **Dependency Array**. This array accepts any variable. Those variables are treated as dependencies and only when any of those variables are changed, the useEffect hook is called (except the first time when useEffect is called when the page is loaded). If we keep the dependency array empty, then useEffect will only be called once, at loading the page.

```
const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState(restaurantList);
    useEffect(() => { console.log("UseEffect called") }, []);
    return (
        <>···
        </>
    )
};
```



No, matter how many times I press Search or input anything, the console remains same.

Now, if we include searchText state variable inside the dependency array, then, when that variable changes, useEffect will be called everytime we input anything. But it won't be called if we just click the Search button, because clicking it does not change that variable.

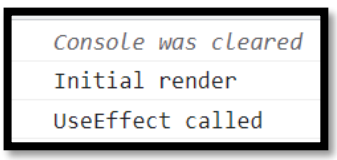
If we include restaurants state variable inside that array, useEffect will be called everytime we click on search button. Even if we did not input anything in the input box, pressing search will automatically call the setRestaurants() function, which assigns some value to the restaurants (even if the new value is same as the previous one). But now, useEffect won't be called when we input anything because value of restaurants isn't changed then.

```
useEffect(() => { console.log("UseEffect called") }, [searchText]);
```

```
useEffect(() => { console.log("UseEffect called") }, [restaurants]);
```

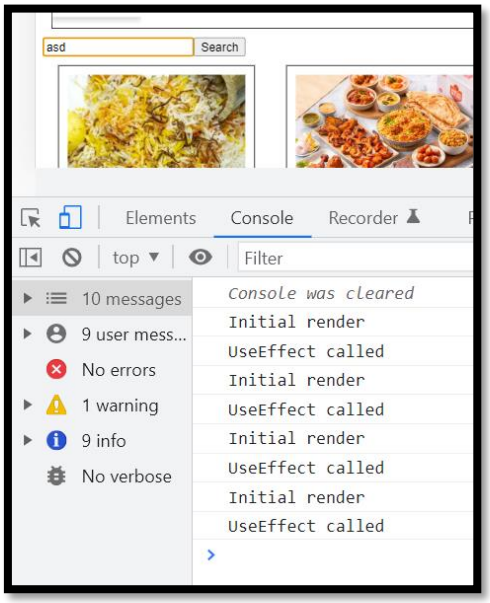
Example 1 :- useEffect() will always be called after initial render and before that. When we run the code below, even though useEffect() is defined first and the console.log("Initial render") after that, the latter will get printed 1st.

```
const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState(restaurantList);
    useEffect(() => { console.log("UseEffect called") }, [searchText]);
    console.log("Initial render");
    return (
        <>···
        </>
    )
};
```



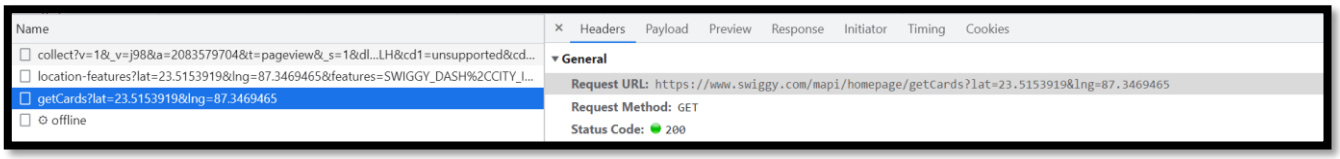
This is the output when we first load the page i.e. when the Body is 1st rendered.

Also, everytime we change the searchText, our Body component is re-rendered and as such “Initial render” is printed 1st and then “UseEffect called”.



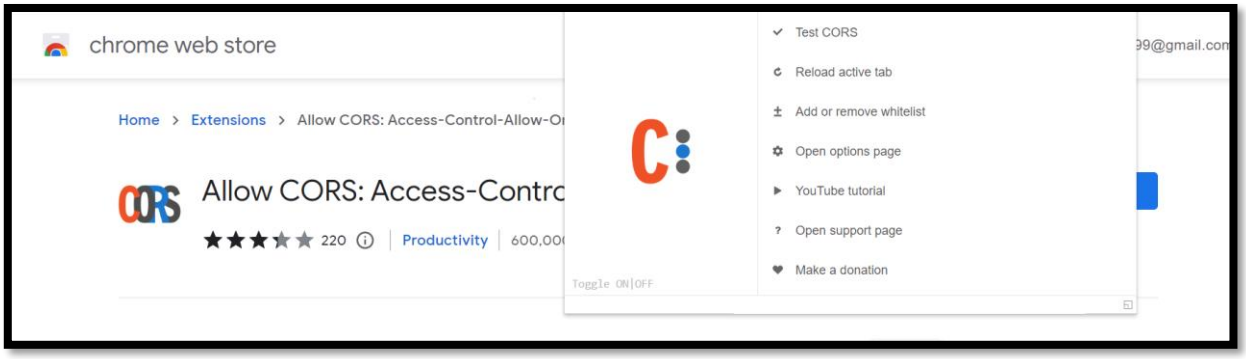
Now, we know that we have to make an API call after the page loads and the initial UI is rendered. So we can make an API call inside the callback function of useEffect() and make the dependency array empty, so that API call is made only when loading the page.

I am using the Swiggy API from here




```
const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState(restaurantList);

    useEffect(() => {
        getRestaurants();
    }, []);
    async function getRestaurants() {
        const data = await fetch("https://www.swiggy.com/mapi/homepa
        const json = await data.json();
        console.log(json);
    }
    console.log("Initial render");

    return (
        <> ···
        </>
    )
};
```

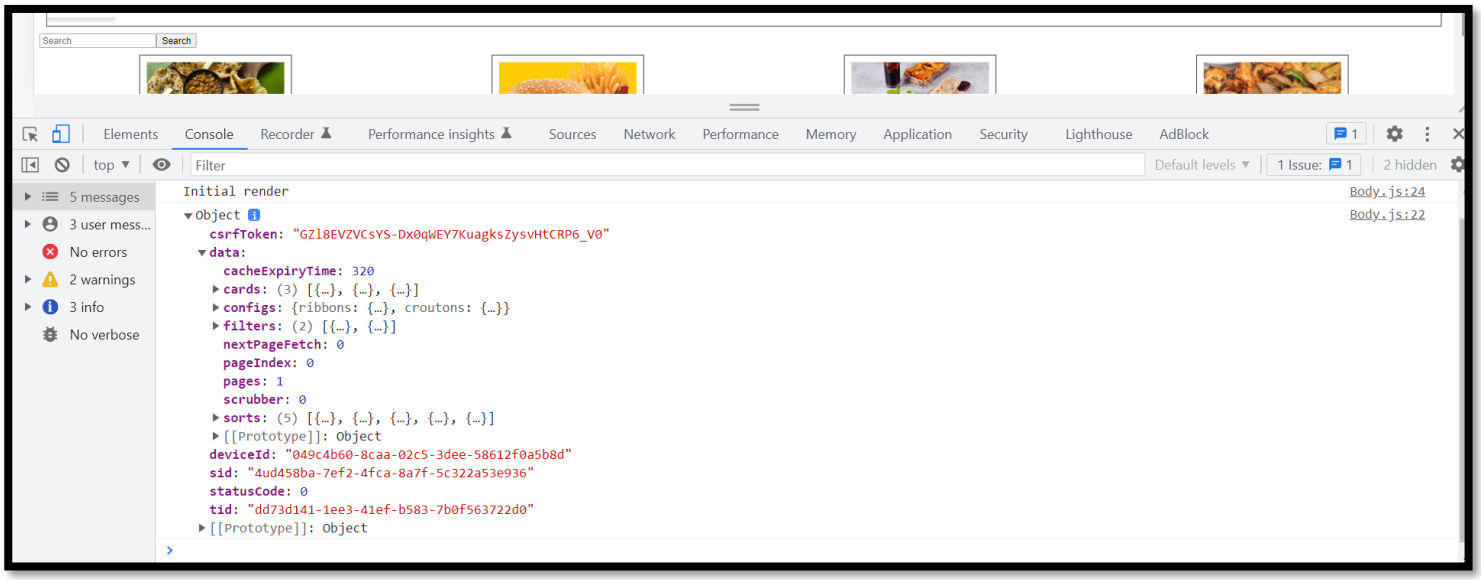
Now, if we run the app, we will be getting a TypeError: Failed to Fetch. This is because Swiggy is not letting us make a call to their API directly. To overcome this, we need a plugin/extension for CORS



To activate the CORS, we have to click on the “toggle on|off” button that you can see after pressing the extension button. Now, if you refresh the page of our app, it will be running fine and we would be getting the same response from the Swiggy API that it was getting earlier.

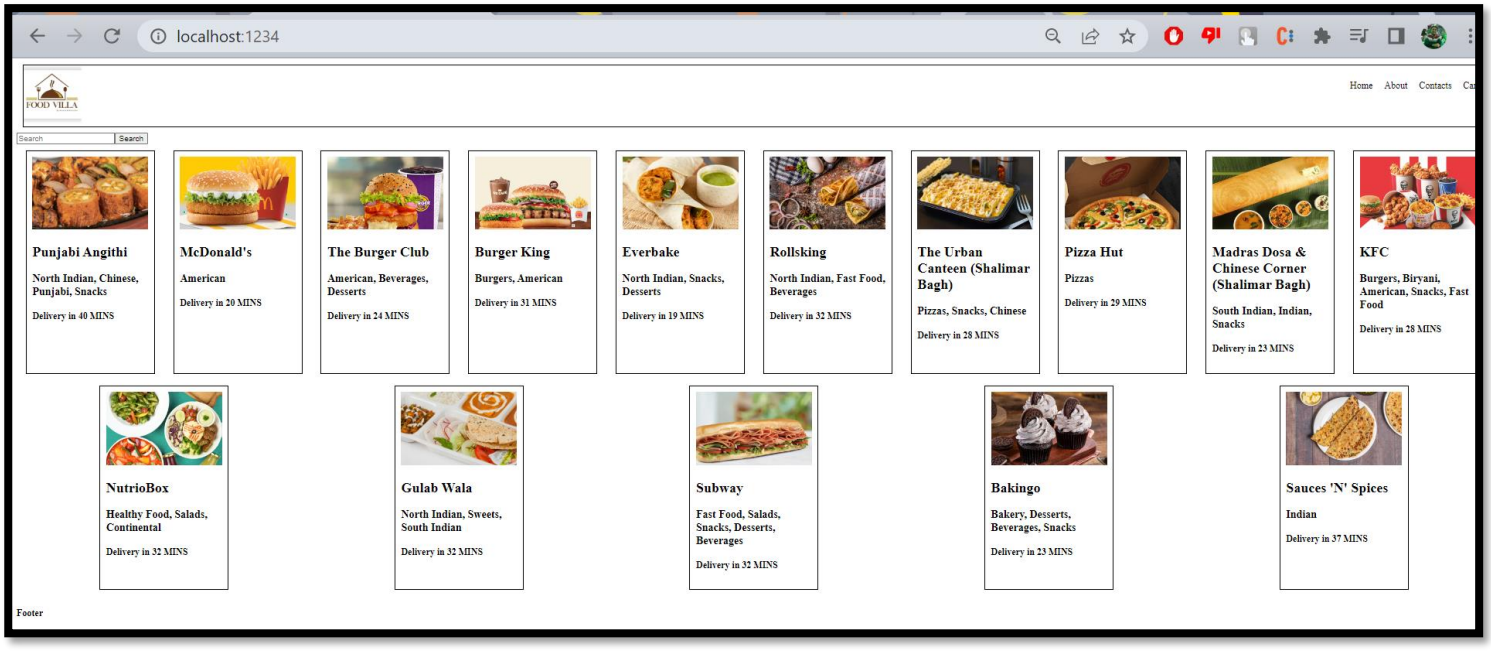
NOTE :- I could not use the API listed above because it was not giving me the restaurants array, so I used another Swiggy API

Now, after I use CORS and the new API, my json will return the same data that swiggy was using :-



Now, we have all the restaurants from this API too and to use it we can call the setRestaurants() with the API data :-

```
import RestaurantCard from "./RestaurantCard";
import restaurantList from "../config/restaurantList";
import { useState, useEffect } from "react";

function filterData(searchText, restaurants) {…
}

const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState(restaurantList);

    useEffect(() => {
        getRestaurants();
    }, []);
    async function getRestaurants() {
        const data = await fetch("https://www.swiggy.com/dapi/restaurants/list/v5?lat=28.717111&lng=77.157598&
        page_type=DESKTOP_WEB_LISTING");
        const json = await data.json();
        console.log(json);
        setRestaurants(json?.data?.cards[2]?.data?.data?.cards);
    }
    console.log("Initial render");

    return (
        <>…
        </>
    )
};
```



And in the console, we get :-

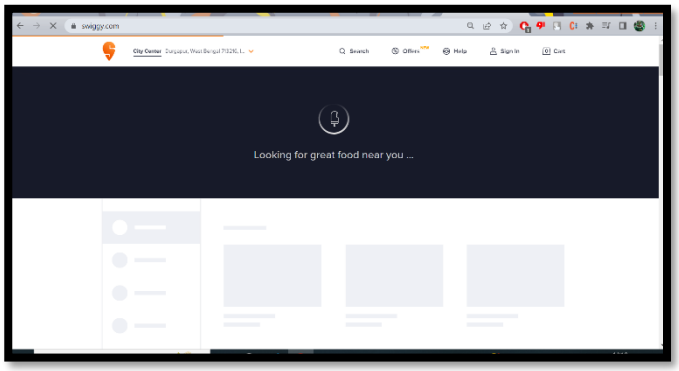
```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools   react-dom.development.js:29840
Initial render                                                                                              Body.js:25
                                                                                                            Body.js:22
▶ {statusCode: 0, data: {…}, tid: '4dd7e3b2-98cd-4035-880c-a8f57802a602', sid: '4udca4a9-bf5b-438c-8ae1-6c3057a9a598', deviceId: '6576315a-e3b2-ec7c-826d-850dd745595
  2', …}
Initial render                                                                                              Body.js:25
>
```

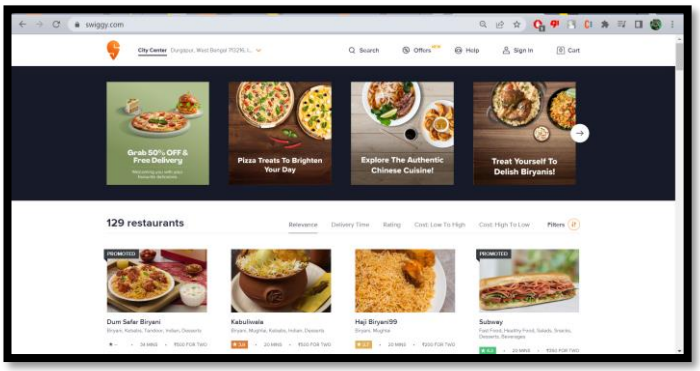
So, when we first load the page, our Body component gets rendered for the 1st time and that's why we get the 1st console log of "Initial render". In the initial render, the value of restaurants is restaurantList. After this rendering is completed, useEffect() is called, which in turn, calls the getRestaurants() function and there we change the value of restaurants to the array we get from API, as well as console log the json object we get from the API call. Once the state variable is re-initialised using the setRestaurants() function, the Body component gets re-rendered (i.e. our UI gets updated) and that's why we see the 3rd console log of "Initial render". But this time after the rendering, useEffect is not called again because of the empty Dependency Array.

Also, you will see that the 1st "Initial render" output gets printed almost immediately, whereas the other 2 output takes some time. And during the time between the 1st console log and the 2nd console log, we seeing the old UI with the data we had manually given.

We don't want that, that's why we remove the restaurantList variable as the initial value of restaurant and just keep it empty. However, that means, till our new UI gets updated with our API data, we will be getting an almost blank page like :-

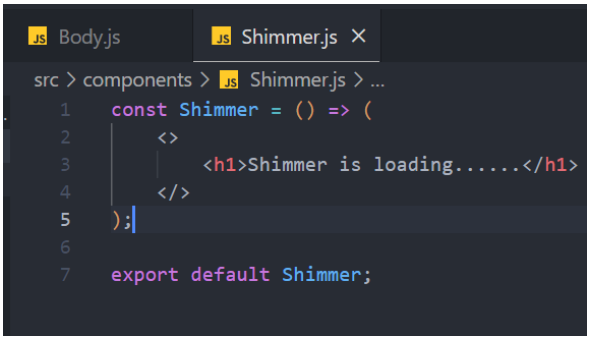
So, in industry, instead of doing this, the developers show an UI in the webpage initially which represents the skeleton of the app's actual UI. It acts like a place holder for the data that has not been loaded yet. Ex :- When Swiggy loads, we first get to see the 1st UI and then when all the data is loaded, we get the whole UI (2nd image)





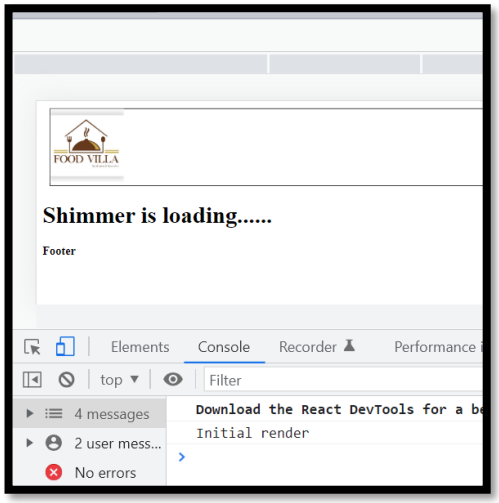
This is called the **Shimmer effect** :- They are loading indicators used when fetching data from a data source that can either be remote or local. It paints a view that may be similar to the actual data to be rendered on the screen when the data is available. So, before we render the actual data UI of our app, we render the Shimmer UI.

Conditional Rendering :- This means we will render a component only when a certain condition is met. We can use this for implementing Shimmer UI by saying that if the data is not fetched yet, we will be showing the Shimmer UI , else the actual data UI. For this, we can make a separate component for ShimmerUI like :-



And we can use this component using conditional rendering in the Body component by :-

```
import Shimmer from "./Shimmer.js";

function filterData(searchText, restaurants) {···
}

const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [restaurants, setRestaurants] = useState([]);

    useEffect(() => {···
    }, []);
    async function getRestaurants() {···
    }
    console.log("Initial render");

    return (restaurants.length === 0) ? (
        <Shimmer />
    ) : (
        <>···
        </>
    )
};

export default Body;
```



So, when the data is not yet fetched, we get the Skimmer UI

Now, you will see that the search button is not working properly. Because earlier my filterData() was :-

```
function filterData(searchText, restaurants) {
    const filterData = restaurantList.filter(
        (restaurant) => restaurant.info.name.toLowerCase().includes(searchText.toLowerCase())
    );
    return filterData;
}
```

So, here we were filtering from the restaurantList array, which does not have all the restaurants objects from the API that we get. So, even though we are showing the API data when our page gets loaded, during search we are filtering from the hardcoded data restaurantList.

So, one might say :- let's change the restaurantList to restaurants state variable. But again, this will work fine for the 1st search but not for the consecutive searches because after 1 search, the restaurants gets updated to the filteredData and it does not hold all the API data anymore.

So, again we need to state variables :- 1 variable to store the filtered data (filteredRestaurant) and the other to store the list of all restaurants from the API (allRestaurants).

Whenever we are filtering, we need to filter from allRestaurants variable and that value will be stored in filteredRestaurant.

Now, in the UI we will only show the filteredRestaurant. That's why initially in the useEffect(), we have to initialise both the variables with all the restaurants. Then we will only only render the data UI component if length of allRestaurants array is not 0. So, in the ternary operation, replace (restaurants.length===0) with (allRestaurants.length===0). Also, in the filterData() function we have to pass allRestaurants instead of restaurants, so that we can perform filter operation on all restaurants.

After the small remaining necessary changes we will get the desired results.

However, there will be a problem :- when we are searching for a restaurant that is not there in our API, then our app will be showing "Shimmer is loading….".

But, we know that it is not because of the data being unable to load, rather, there not being any restaurants matching the user's filter. To, handle that we use **Early Returns :- a component is returned early if a criteria is met.**

Here, if we see that the filteredData (i.e the filteredRestaurant) is an empty array, we will return a component saying that we could not find any restaurant like :- (See line 30 of the 1st image in next page).

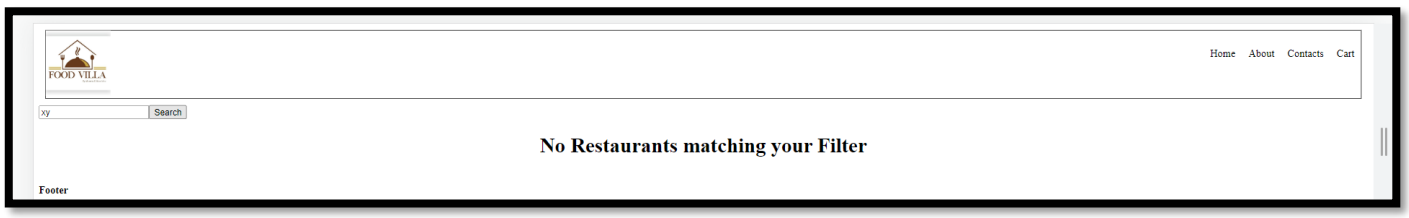
However, this gives to another problem :- Whenever we first load the page, the filteredRestaurant variable is also empty and thus instead of showing "Skimmer is loading….", it shows "No Restaurants matching your filter". And there's also an issue that every time, there is early return, we are losing our search bar, because intead of the component with search bar, the early return component gets rendered. So to overcome these problems, instead of writing the "Early Returns" code before the whole return component, we should write in inside the div with the map function, so that we can have our search bar if no restaurants are found and the skimmer component when data is not fetched both.

```
    useEffect(() => {
        getRestaurants();
    }, []);
    async function getRestaurants() {…
    }
    console.log("Initial render");

    if (filteredRestaurants.length === 0)
        return (<h1>No Restaurants Found</h1>);

    return (allRestaurants.length === 0) ? (…
    ) :
        (
            <>…
            </>
        )
```

So, if we do the below thing :-

```
const Body = () => {
    let [searchText, setSearchText] = useState("");
    const [filteredRestaurants, setFilteredRestaurants] = useState([]);
    const [allRestaurants, setAllRestaurants] = useState([]);

    useEffect(() => {
        getRestaurants();
    }, []);
    async function getRestaurants() {…
    }
    console.log("Initial render");

    return (allRestaurants.length === 0) ? (…
    ) :
        (
            <>
                <div className="search-container">
                    <input…
                    </input>
                    <button className="search-btn" onClick={() => {…
                    }}>Search</button>
                </div>
                <div className="restaurantList">
                    {
                        (filteredRestaurants.length === 0) ? (<h1>No Restaurants matching your Filter</h1>) :
                            filteredRestaurants.map((restaurant) => {
                                return <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
                            })
                    }
                </div>
            </>
        )
};
```

We get the “Skimmer is loading….” Output when data is not fetched yet and when no Restaurants are found, we get :-



#NOTE :-

In JSX, we said that we can write JS inside JSX by wrapping it inside “ {} “. However, the actual truth is :- we can only write JS expressions (those piece of JS code that produces a value -> if you type that piece of code in browser console and hit enter, if it produces a value, then it’s an expression, else it’s a statement) and not JS statements. Refer to these for further read :- Link1, Link2, Link3, Link4, Link5 (In this link, there’s a link to another website :- Link6, check how to use this website from link5 too). Also Link7 (Official docs). Now, we can use multiple expressions as a singe expression like :-

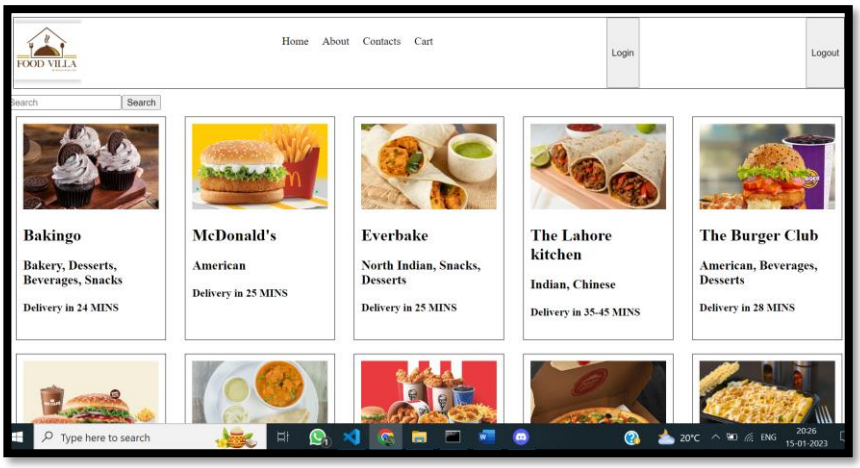
```
            </div>
            {
                ((a = 5), (console.log(a)))
            }
            <button>Login</button>
            <button>Logout</button>
        </div>
```

Login/ Logout Button

We will create these buttons in the header section like :-

```
const Header = () => (
    <div className="header">
        <Title />
        <div className="nav-items">
            <ul> ···
            </ul>
        </div>
        <button>Login</button>
        <button>Logout</button>
    </div>
)
```

And the UI for this will be :-



Now, we don't want to display both the buttons at the same time. First we should display the login button, and when the user is logged in, we should display the logout button. For practise purposes, we will just use a state variable isLoggedIn which has a default value false (i.e. the user is not logged in by default), with only the LogIn button shown in UI and on clicking the Login button, the value of he state variable isLoggedIn whould be changed to true, with only the Logout button shown.

```
const Header = () => {
    const [isLoggedIn, setIsLoggedIn] = useState(false);
    return (
        <div className="header">
            <Title />
            <div className="nav-items">
                <ul> ···
                </ul>
            </div>
            {
                (!isLoggedIn) ?
                    (<button onClick={() => { setIsLoggedIn(true) }}>Login</button>)
                    : (<button onClick={() => { setIsLoggedIn(false) }}>Logout</button>)
            }
        </div>
    )
};
```