Chapter 6 – Exploring the World

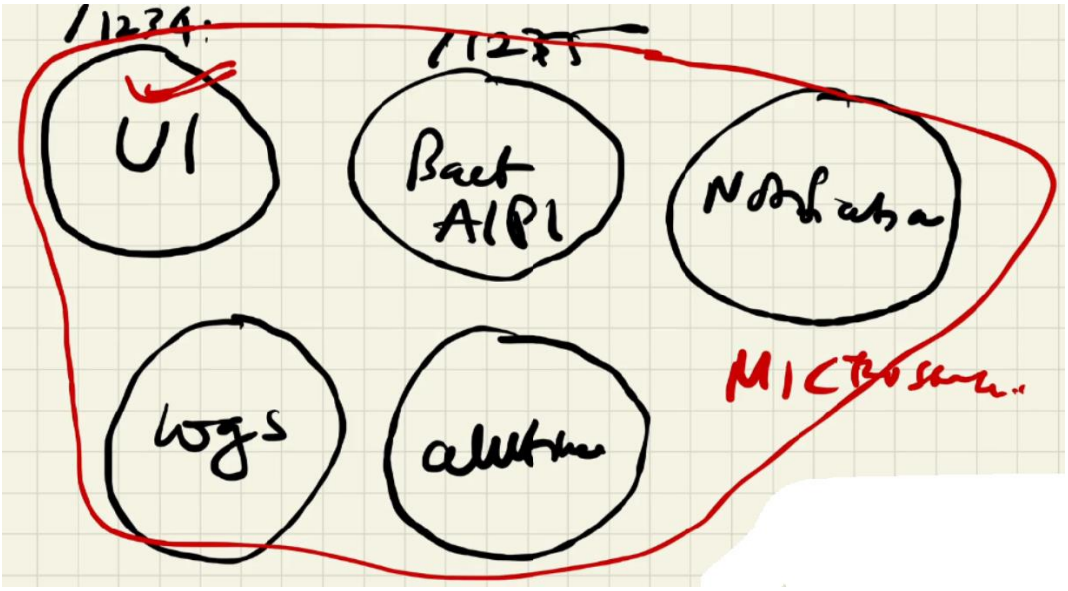
Before we explore external worlds, we should know some design patterns followed in software industry

What is Microservice?

Microservices is an architectural style that structures an application as a collection of services that are Independently deployable, loosely coupled and owned by a small Team. These services are communicated to each other over well-designed APIs.

Now newer start-ups are moving towards micro services. When we have to build large scale application only one react project is not enough. Instead of having just one project, now we have small different projects. We have UI project, Backend API project, Notification Project, Logs Project, Authentication project and much more and these projects are hosted in different ports. This architecture is known as Micro Services Architecture.

Advantages – Easier to test, if frontend related changes are made, only deploy the UI project. Hence, it's easy to maintain. Every team can have separate responsibility. The backend team can clone one project and they don't have to worry about UI project. Same applies to another team. This is like separation of concern. This comes under single responsibility principle SRP.



Time to explore the world. We are going to fetch some real time data from an API that our application does not know.

How to make an API call?

- In JavaScript we call API using fetch ()
- In react we call API inside UseEffect () hook. (discussed later)

Ways of API call in react

There are two ways we can call API in react

Way 1: We Load the website **=>** make the API call (300ms) **=>** render the page(200ms)

Way 2: We Load the website **=>** render the initial page (100ms) **=>** make the API call (300ms) **=>** Update the UI.

In way 1, page will be available to us in (300 + 200 = **500**) ms (milli seconds)

In way 2, Page will be available to us in **100ms** which is lesser than the first approach

Way 2 is a good way of calling API because of the user Experience.

Best place to call an API in React

We can't just call API anywhere within our component. Because every time our states and props change, every time UI is updated, component rerenders. Every time our component rerenders, API gets called which is a big performance hit.

What we want is on initial page load we want our API to be called just once. To make this functionality happen React gives us access to the second most important hook known as useEffect hook.

What is useEffect Hook?

useEffect is a hook given by react which runs after a component renders and rerenders in DOM provided the hook must be placed inside that component.

* useEffect hook expects two parameters. First parameter is a callback function and second one is a dependency array.

* The callback function will not be called immediately, but it gets called when useEffect wants it to be called.

* When a component renders or rerenders, useEffect calls its callback which is the default behaviour of useEffect hook.

* We can control this action by providing a dependency array. If dependency array is empty useEffect calls its callback just once which is after component's initial render but when the component rerenders useEffect never calls its call back.

* Once we provide a state variable to the dependency array, useEffect calls its call back after component's initial render and subsequent renders in response to the state variable change.

* We make API call inside the callback function of useEffect and provide an empty dependency array to the hook so that API will be called just once.

* The dependency array is optional. If we don't provide the dependency array useEffect will not be dependent on anything. So, every time component is rendered useEffect's default behaviour kicks in.

When does our component renders and rerenders?

When our page loads for the first-time component renders. For every change in states and props component rerenders.

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

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

  async function getRestaurants() {
    const data = await fetch(
      "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9715987&lng=77.5945627&is-seo-homepage-enabled=true&
      page_type=DESKTOP_WEB_LISTING"
    );
    const jsonData = await data.json();
    const restaurantListData =
      jsonData?.data?.cards[5]?.card?.card?.gridElements?.infoWithStyle
        ?.restaurants;

    /* Body component rerenders when below line get executed because here restaurants state is changed from [] to
    [<resList>] */
    setRestaurants(restaurantListData);
  }
```

In this code, API will be called once after the component renders for the first time because in useEffect hook, dependency array is empty.

What is Shimmer UI?

Shimmer UI is a representation of a fake page with shimmering effect. A shimmer effect is a visual effect used in UI design where an animated shiny light passes over an object or text giving the impression that the object is shimmering or gleaming.

In older days we used to show loader on the screen until we fetch the data from an API and display it in UI. Shimmer UI works wonders giving the end user a much better UI/UX experience.

In below figure, End user can anticipate there are cards which will be loading in Shimmer UI and user gets the card when data is fetched from the API.

Conditional Rendering in React

Rendering components based on a given condition check is known as Conditional Rendering.

Syntax –

return (condition) ? <component1> : <component2>

If condition is true component1 is returned and rendered in UI, unless component2 is returned and rendered in UI

In below example we are showing shimmer to the UI when restaurant list data is not available.

As we know useEffect will be called after a component renders and rerenders, in our case useEffect will be called just once as we have provided an empty dependency array.

When the component is rendering, restaurants state variable is initialised with empty array [] and as per conditional rendering logic we are showing shimmer in the UI.

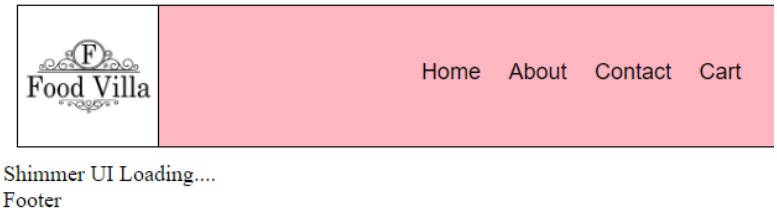
When component's initial render completes, useEffect gets called which in turn invokes the callback function that calls the API and get the restaurant data. Until the data is fetched the shimmer is ON in the UI. when we get back the data, setRestaurants updates restaurants with the updated data.

When restaurants state is changed Body component rerenders and as per conditional rendering logic restaurants state variable will have data this time which displays list of restaurants in the UI.

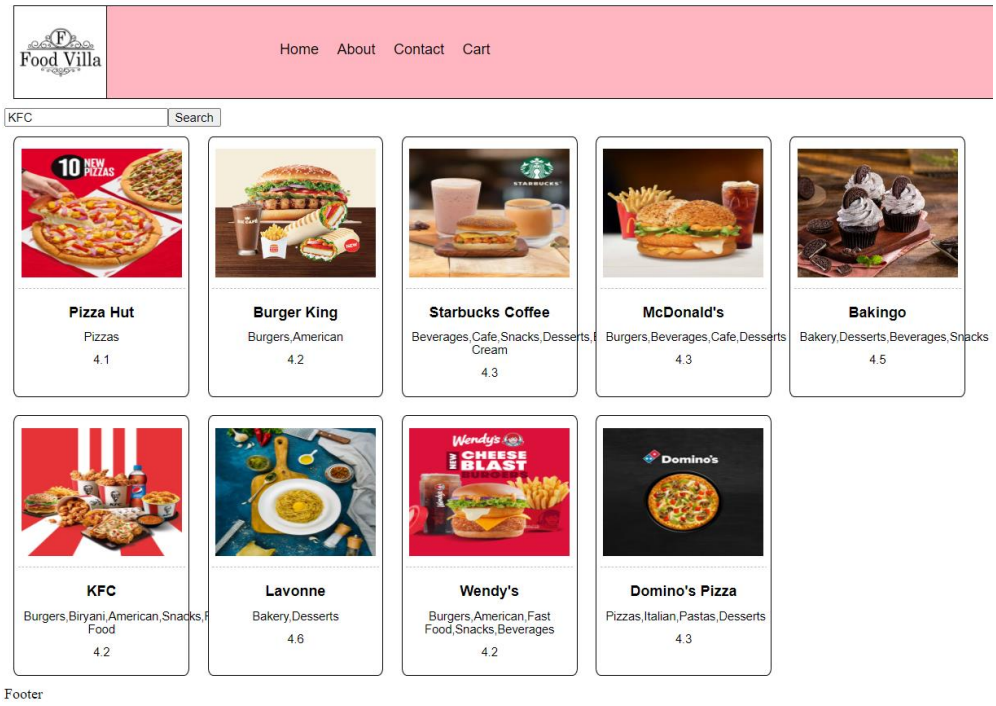
```
// Conditional Rendering -
// If restaurant list is empty show shimmer UI.
// If restaurant list is not empty, show the actual UI filled with restaurants data.

return restaurants?.length === 0 ? (
  <Shimmer />
) : (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchText}
        onChange={(e) => setSearchText(e.target.value)}
      />
      <button ···
      </button>
    </div>
    <div className="restaurant-List">
      {restaurants?.map((restaurant, index) => {
        return <RestaurantCards data={restaurant} key={index} />;
      })}
    </div>
  </>
);
};
```

When restaurants data is not available – (We have hardcoded shimmer effect for now. We will make our own shimmer in next chapter)



When restaurants data is available -



In last chapter we have seen how the search functionality was working, it was doing a single search and on subsequent search the functionality was not working.

Why search is behaving the way it should not behave?

When we typed some text in the textbox and clicked on search button, filterData function got invoked which returned a list of filtered restaurants data. Once we received the filtered data we updated the restaurants state variable via setRestaurants(filteredData).

In response to state change the Body Component rerendered which displayed the filtered restaurant card component in UI.

At this moment the restaurants state variable is filled with filtered data. Thus, on next search, whatever we typed in the text box was searched across these stored filterData. if the restaurant was found we could have got the restaurant card component rendered in the UI but we got a blank page meaning the restaurant was not found in the filtered data.

```
<button
  className="search-btn"
  onClick={() => {
    const filteredData = filterData(searchText, restaurants);
    setRestaurants(filteredData);
  }}
>
  Search
</button>
```

To fix this issue, we need this search operation to be performed over list of all restaurants, not list of filtered restaurants. To do that we need two state variables.one variable is for list of all restaurants and another state variable is list of filtered restaurants.

At any point of time we want all restaurants and filtered restaurants. Thus, creating two state variables inside Body component.

Body.js

src > components > Body.js > Body > getRestaurants

```
1   import RestaurantCards from "./RestaurantCards";
2   import { useState, useEffect } from "react";
3   import Shimmer from "./shimmer";
4
5   function filterData(searchText, allRestaurants) {…
12  }
13
14  const Body = () => {
15    const [filteredRestaurants, setFilteredRestaurants] = useState([]);
16    const [allRestaurants, setAllRestaurants] = useState([]);
17    const [searchText, setSearchText] = useState("KFC");
18    useEffect(() => {…
20    }, []);
21    async function getRestaurants() {
22      const data = await fetch(
23        "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9715987&lng=77.5945627&is-seo-homepage-enabled=true&
          page_type=DESKTOP_WEB_LISTING"
24      );
25      const jsonData = await data.json();
26      const restaurantListData =
27        jsonData?.data?.cards[5]?.card?.card?.gridElements?.infoWithStyle
28          ?.restaurants;
29      setAllRestaurants(restaurantListData);
30      setFilteredRestaurants(restaurantListData);
31    }
32
33    if (!allRestaurants) return null; // Early return
34
35    if (filteredRestaurants?.length === 0)
36      return <h1>No restaurant match your filter</h1>;
37
38    return allRestaurants.length === 0 ? (…
40    ) : (…
66    );
67  };
68
69  export default Body;
```

filteredRestaurants state variable stores list of filtered restaurants and displays them on UI when filter is applied. Initially filteredRestaurants stores list of all restaurants when filter was not applied.

allRestaurants state variable stores list of all restaurants

When the Body component renders for the first time allRestaurants and filteredRestaurants state variables are empty which we are then filled up with data fetched from API. (Line 29 and Line 30). We are creating two copies of lists of restaurants. One for storing and another one for updating.

```
<button
  className="search-btn"
  onClick={() => {
    const filteredData = filterData(searchText, allRestaurants);
    setFilteredRestaurants(filteredData);
  }}
>
  Search
</button>
```

Now if we click on search button, filterData function is invoked which takes searchText and allRestaurants. This allRestaurants state variable always stores list of all restaurants and will not be changed in our application.

Earlier we were applying filter logic to a state variable which was meant for storing and rendering data in UI. This time we are applying filter logic to a state variable allRestaurants, doing filter operation on it and getting the filtered data using which we are updating a new state variable filteredRestaurants.

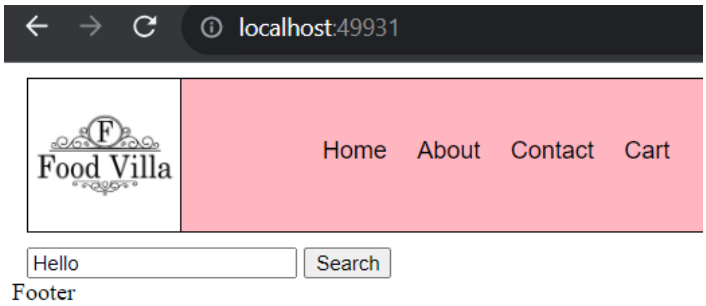
allRestaurants acts a data storage whereas filteredRestaurants acts as data provider

Filter Logic -

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

Now the search function behaves the way it should behave.

But we have another problem in our app. If the filtered data does not exist we see blank in UI.



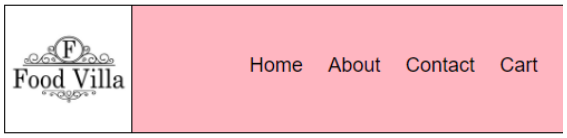
Let's handle this.

```
    if (filteredRestaurants?.length === 0)
      return <h1>No restaurant match your filter</h1>;

  return allRestaurants.length === 0 ? (···
  ) : (···
  );
};

export default Body;
```

Now for invalid search operation we get the notification that No restaurant matches your filter. But it caused another problem, our search bar is gone. We will fix this in upcoming chapters.



No restaurant match your filter

Footer

Log-in Log-out Toggle Feature-

```
const Header = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  return (
    <div className="header">
      <Title />
      <div className="nav-items">
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
      </div>
      {isLoggedIn ? (
        <button onClick={() => setIsLoggedIn(false)}>Log out</button>
      ) : (
        <button onClick={() => setIsLoggedIn(true)}>Log in</button>
      )}
    </div>
  );
};

export default Header;
```

When we click on Login button, Logout button is displayed and vice versa

What is Early Return? Why do we use it?

Early return means returning from a component early.

We use early return because we need to stop rendering a component when data is not available. If data is not available and we try to render a component based on some condition check, the condition fails and we get unexpected error in UI.

In below example when we run our app, during initial page load allRestaurants state variable is undefined because the data is not available to us early and we are showing shimmer UI when allRestaurants array is empty.

Body.js | Header.js

src > components > Body.js > Body

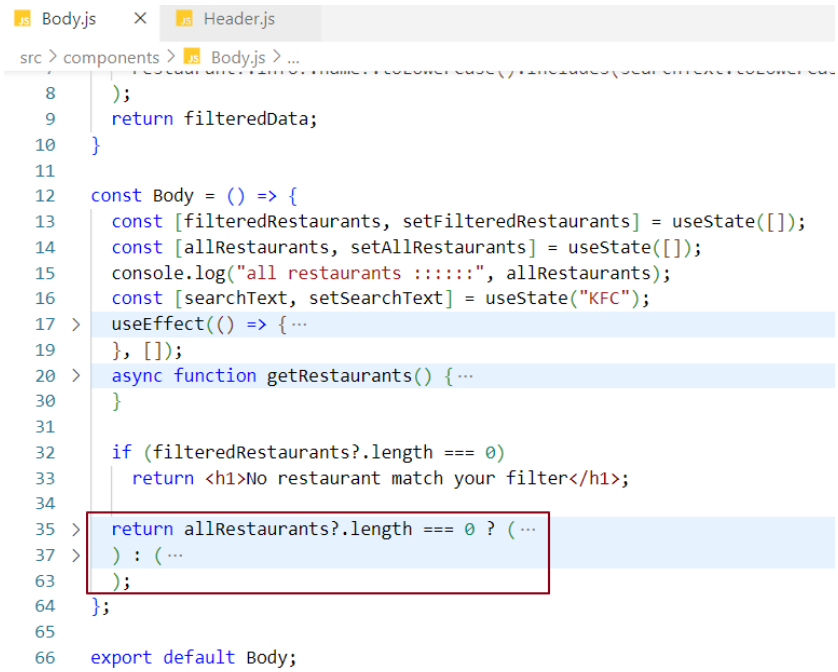
```
1   import RestaurantCards from "./RestaurantCards";
2   import { useState, useEffect } from "react";
3   import Shimmer from "./shimmer";
4
5   function filterData(searchText, allRestaurants) {
6     const filteredData = allRestaurants.filter((restaurant) =>
7       restaurant?.info?.name?.toLowerCase().includes(searchText.toLowerCase())
8     );
9     return filteredData;
10  }
11
12  const Body = () => {
13    const [filteredRestaurants, setFilteredRestaurants] = useState([]);
14    const [allRestaurants, setAllRestaurants] = useState([]);
15    console.log("all restaurants ::::::", allRestaurants);
16    const [searchText, setSearchText] = useState("KFC");
17  > useEffect(() => {···
19    }, []);
20  > async function getRestaurants() {···
30    }
31
32    if (filteredRestaurants?.length === 0)
33      return <h1>No restaurant match your filter</h1>;
34
35  > return allRestaurants.length === 0 ? (···
37  > ) : (···
63    );
64  };
65
66  export default Body;
```

```
all restaurants :::::: undefined
```

But allRestaurants is undefined and we can't get access to undefined length property. Thus, in UI we get below error

There are two ways to avoid such errors.

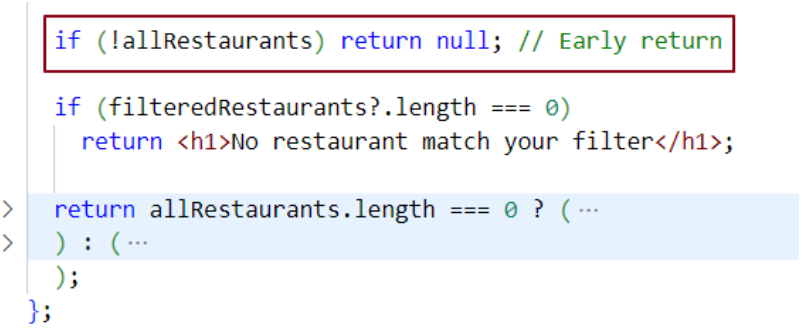
Way 1: Optional chaining



Way 2: Early return

We stop rendering the shimmer component until allRestaurants is not undefined. We do early return.

In our case before rendering the shimmer component, we are doing early return so that if allRestaurants is not available or undefined, return null or a piece of valid JSX.This way we can stop the shimmer component being rendered early on UI.



Q: What is the difference between JS expression and JS statement?

JS expression -

A JS expression returns a value that we use in the application.

example:

1 + 2 // expresses 3
"foo".toUpperCase() // expresses 'FOO'
console.log(2) // logs '2'
isTrue ? true : false // returns us a true or false value based on isTrue value

JS statement -

A JS statement, does not return a value.

example:

let x; // variable declaration
if () { } // if condition

If we want to use JS expression in JSX, we have to wrap in {/* expression slot */} and if we want to use JS statement in JSX, we have to wrap in {(/* statement slot */)};

Some important points

* DOM updates happen via react DOM library.

* Diffing algorithm is written inside react core library.