Chapter 9 – Optimising App

Why to use hook in React when we have normal JavaScript function?

Because in hooks we can define state variables so that react can keep track of these variables and triggers reconciliation on every state change. When it comes to a normal JavaScript function we can't define state variables inside the function so that React cannot trigger reconciliation process.

Why should we build our own hook? Why custom hook?

* Reusability - Hooks are JavaScript functions. In a function we write logic once and use it as many times as per the need. This is code reusability. Just like functions we can also reuse React hooks.

* Readability: By looking at the hook we get to know why the hook is built for. We provide meaningful name to the custom hooks so that we get to know what the hook should be doing in our code.

* Modularity: Each react component should have a single responsibility which is to render the data in UI. We should separate functional logics away from the component and offloads this responsibility to a custom hook or a helper function and use them whenever required. This is called modularity meaning breaking down the code into smaller pieces, each having their own responsibility.

* Testability: We break down larger piece of code into smaller chunks that we store in utilities files which enables us to write more test cases.

* A component should focus solely on a single task for which it is built for even if it has the capability to carry out multiple tasks.

Assuming we have a component that makes an API call to fetch some data and once the data is available, component renders the data in UI. Two tasks are being performed by the component, API call and data rendering. We should offload API call responsibility to a custom hook so that our component will focus only on one specific task which is data rendering in UI.

We have written the code in such a way that it is breaking Modularity.

We have written Filter Logic inside Body component. As per SRP, Body component should only render data in UI. Body component should not worry about how the data is Filtered. We separated this logic out from Body component and placed it inside a helper module so that when Body component needs this module functionality, it gets it from the import.

Body.js helper.js

src > utils > helper.js > filterData

```
1  export function filterData(searchText, allRestaurants) {
2    console.log("Search Text = ", searchText);
3    console.log("All restaurants", allRestaurants);
4    const filteredData = allRestaurants.filter((restaurant) =>
5      restaurant?.info?.name?.toLowerCase().includes(searchText.toLowerCase())
6    );
7    return filteredData;
8  }
```

utils

helper.js

Body.js × helper.js

src > components > Body.js > Body > filteredRestaurants.map() callback

```
1   import RestaurantCards from "./RestaurantCards";
2   import { useState, useEffect } from "react";
3   import ShimmerUI from "./ShimmerUI.js";
4   import { Link } from "react-router-dom";
5   import { filterData } from "../utils/helper";
6   import useOnline from "../utils/useOnline";
7
8   const Body = () => {
9     const [filteredRestaurants, setFilteredRestaurants] = useState([]);
10    const [allRestaurants, setAllRestaurants] = useState([]);
11    const [searchText, setSearchText] = useState("KFC");
12
13    useEffect(() => {···
15    }, []);
16
17    async function getRestaurants() {···
28    }
29
30    const isOnline = useOnline();
31    if(!isOnline){···
33    }
34
35    if (!allRestaurants) return null;
36
37    return allRestaurants.length === 0 ? (
38      <ShimmerUI />
39    ) : (
40      <>
41        <div className="search-container">
42          <input
43            type="text"
44            className="search-input"
45            placeholder="Search"
46            value={searchText}
47            onChange={(e) => setSearchText(e.target.value)}
48          />
49          <button
50            className="search-btn"
51            onClick={() => {
52              const filteredData = filterData(searchText, allRestaurants);
53              setFilteredRestaurants(filteredData);
54            }}
55          >
56            Search
57          </button>
58        </div>
59        <div className="restaurant-List">
60          {filteredRestaurants?.map((restaurant) => {
61            {
62              console.log(restaurant);
63            }
64            return (
65              <Link to={"/restaurant/" + restaurant.info.id}>
66                <RestaurantCards data={restaurant} key={restaurant.info.id} />;
67              </Link>
68            );
69          })}
70        </div>
71      </>
72    );
73  };
74
75  export default Body;
```

We have written API call logic inside Restaurant Menu component. As per SRP, Restaurant Menu component should only render data in UI. Restaurant Menu component should not worry about how the data is fetched from the API. We separated this logic out from Restaurant Menu component and placed it inside a custom hook in helper module useRestaurant.js so that when Restaurant Menu component needs this module functionality, it gets it from the import.

RestaurantMenu.js | useRestaurant.js | helper.js

src > utils > useRestaurant.js > useRestaurant

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

var Menus = [];

// creating a Custom Hook
const useRestaurant = (id) => {
  const [restaurant, setRestaurant] = useState(null);
  const [menuCategories, setMenuCategories] = useState([]);

  // get Data from API
  useEffect(() => {
    getRestaurantInfo();
  }, []);

  async function getRestaurantInfo() {
    const data = await fetch(
      FETCH_MENU_URL +
        id +
        "&catalog_qa=undefined&
        metaData=%7B%22type%22%3A%22RESTAURANT%22%2C%22data%22%3A%7B%22parentId%22%3A6292%2C%22primaryRes
        taurantId%22%3A29087%2C%22cloudinaryId%22%3A%22dw307jhy1c2t3gmzy5zn%22%2C%22brandId%22%3A6292%2C%
        22dishFamilyId%22%3A%22846613%22%2C%22enabled_flag%22%3A1%7D%2C%22businessCategory%22%3A%22SWIGGY
        _FOOD%22%2C%22displayLabel%22%3A%22Restaurant%22%7D&submitAction=SUGGESTION"
    );
    const json = await data.json();
    const restaurantData = json.data.cards[0].card.card.info;
    setRestaurant(restaurantData);
    const MenuItemsList = await json.data.cards[2].groupedCard.cardGroupMap
      .REGULAR.cards;
    MenuItemsList.slice(1, -2).map((item) => {
      Menus.push(item.card.card.title);
    });
    setMenuCategories(Menus);
  }

  // return restaurant data and menuCategories data
  return [restaurant, menuCategories];
};

export default useRestaurant;
```

RestaurantMenu.js × useRestaurant.js helper.js

src > components > RestaurantMenu.js > default

```
1   import { useEffect, useState } from "react";
2   import { useParams } from "react-router-dom";
3   import { IMG_CDN_URL } from "../constants";
4   import ShimmerUI from "./ShimmerUI";
5   import useRestaurant from "../utils/useRestaurant";
6
7   const RestaurantMenu = () => {
8     const { id } = useParams();
9
10    const [restaurant,menuCategories] =  useRestaurant(id)
11
12    if (!restaurant) {
13      return <ShimmerUI />;
14    }
15
16 >  return ( ···
40    );
41  };
42
43  export default RestaurantMenu;
```

Building Online and Offline features –

If the user has no internet connection then it should show You are Offline check your internet connection else it should show the actual data.

RestaurantMenu.js useOnline.js × useRestaurant.js

src > utils > useOnline.js > ...

```
1   import { useEffect, useState } from "react";
2
3   const useOnline = () => {
4     const [isOnline, setIsOnline] = useState(true);
5
6     useEffect(() => {
7       const handleOnline = () => {
8         setIsOnline(true);
9       };
10      const handleOffline = () => {
11        setIsOnline(false);
12      };
13      window.addEventListener("online", handleOnline);
14      window.addEventListener("offline", handleOffline);
15      // clean-up code
16      return () => {
17        window.removeEventListener("online", handleOnline);
18        window.removeEventListener("offline", handleOffline);
19      };
20    }, []);
21    return isOnline;
22  };
23
24  export default useOnline;
```

Code Explanation -

In JavaScript, window object provides a method called addEventListener where we attach events. In our case we are attaching online and offline events to addEventListener method, also we are providing a callback function as a second argument to addEventListener which gets called when user gets online or offline.

We want this event to be called just once for both online and offline. That's why we have put them inside useEffect hook.

When we navigate from component to component, online and offline event listeners should have been removed but they still exist. It is always a good practice to remove event listeners on components navigation otherwise browser will keep hold to these events which is a big performance hit. In our code we are already handling this via RemoveEventListener during component unload phase.

Body.js

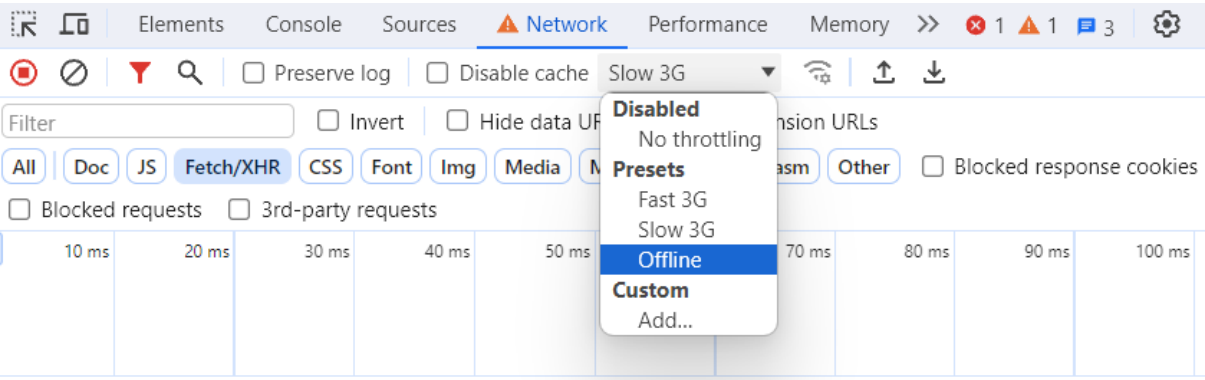
src > components > Body.js > ...

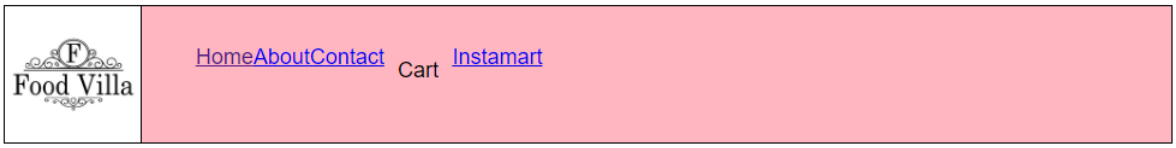
```
1   import RestaurantCards from "./RestaurantCards";
2   import { useState, useEffect } from "react";
3   import ShimmerUI from "./ShimmerUI.js";
4   import { Link } from "react-router-dom";
5   import { filterData } from "../utils/helper";
6   import useOnline from "../utils/useOnline";
7
8   const Body = () => {
9     const [filteredRestaurants, setFilteredRestaurants] = useState([]);
10    const [allRestaurants, setAllRestaurants] = useState([]);
11    const [searchText, setSearchText] = useState("KFC");
12
13  > useEffect(() => {···
15    }, []);
16
17  > async function getRestaurants() {···
28    }
29
30    const isOnline = useOnline();
31    if(!isOnline){
32      return <h1>Offline,please check your internet connection !!</h1>
33    }
34
35    if (!allRestaurants) return null;
36
37  > return allRestaurants.length === 0 ? (···
39  > ) : (···
72    );
73  };
74
75  export default Body;
76
```

Simulating offline online behaviour in chrome browser -

Offline – offline mode

No throttling – online mode





Offline,please check your internet connection !!

Footer

Using useOnline hook in header.js -

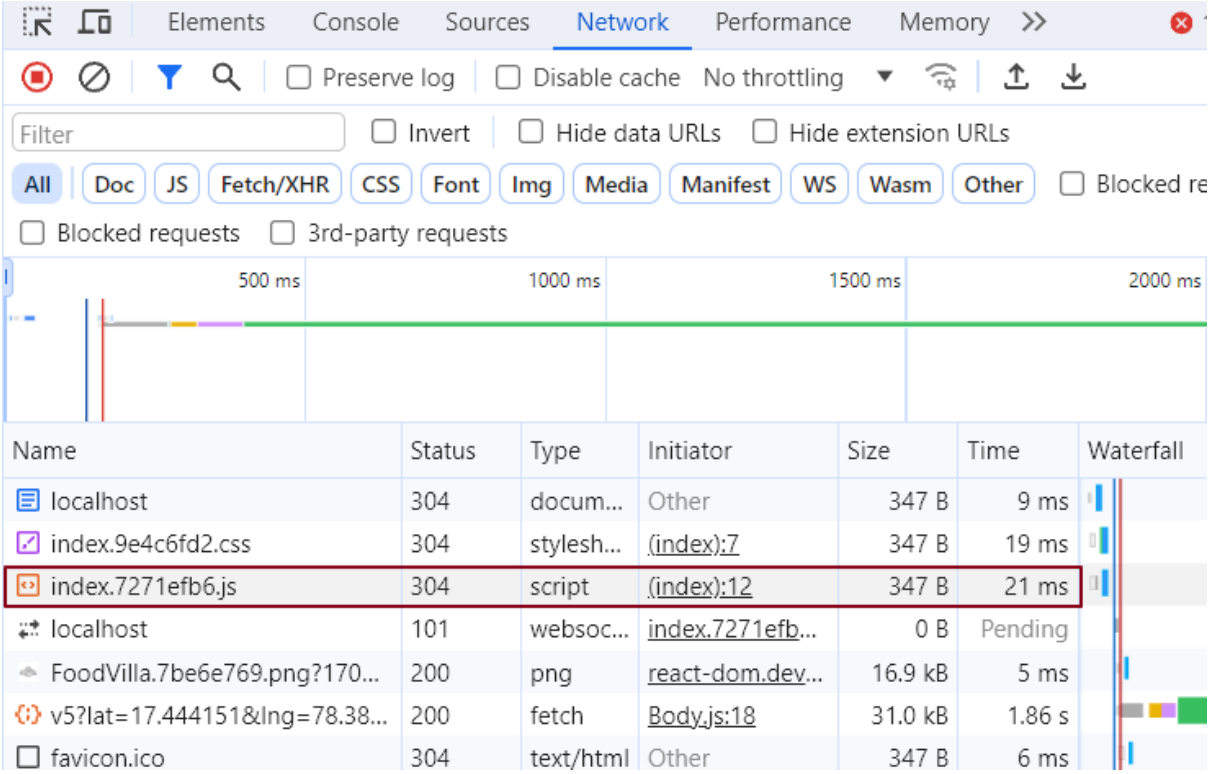
useOnline.js | Header.js | Body.js

src > components > Header.js > ...

```
1   import { useState } from "react";
2   import Logo from "../assets/Img/FoodVilla.png";
3   import { Link } from "react-router-dom";
4   import useOnline from "../utils/useOnline";
5
6   const loggedInUser = () => {
7     return true;
8   };
9
10  const Title = () => (
11    <a href="/">
12      <img className="logo" src={Logo} alt="" />
13    </a>
14  );
15
16  const Header = () => {
17    const [isLoggedIn, setIsLoggedIn] = useState(false);
18    const isOnline = useOnline();
19    return (
20      <div className="header">
21        <Title />
22  >     <div className="nav-items">···
38        </div>
39        <h1>{isOnline ? "Online" : "Offline"}</h1>
40        {isLoggedIn ? (
41          <button onClick={() => setIsLoggedIn(false)}>Log out</button>
42        ) : (
43          <button onClick={() => setIsLoggedIn(true)}>Log in</button>
44        )}
45      </div>
46    );
47  };
48
49  export default Header;
```

With the help of custom hook, we can reuse the code. We can achieve code reusability.

In our application parcel created only one JavaScript file where all of our code is bundled and minified together.



The size of the index file bundle is less in production build.

In large scale application, there are thousands of components and if we bundle them all together in one file the application will become very slow. Because when the page loads for the first time, all components from the bundle will be loaded at once. To make the application scalable, performant and efficient we need to chunk the big bundle into individual mini bundles and load these bundles on certain conditions.

What is bundling? Who does bundle process?

Bundling is a process of merging imported files or code dependencies into a single optimised & minified file. Bundler such as parcel, web pack does the bundling.

What is Chunking / Code Splitting / Dynamic bundling / Lazy Loading / On demand Loading / Dynamic Import?

If we build a large-scale production ready application, there will be multiple components and bundling them all together in a single JS file will take so much time to render in UI. Therefore, we should break the entire production ready code into smaller chunks. This concept is known as Chunking or code splitting.

Is bundling good?

Bundling is good, but to a certain extent.

We don't have to bundle all components code into a single file and load the file in UI. Instead, we should make logical bundles. This means we should code in such a way that bundle corresponding to a specific use case will only get loaded in UI. Let's try to understand this with an example.

* Make My Trip (MMT) is an application that provides many services but primarily people use it for booking flights. So, when they open MMT they land on flights page because from developer perspective we load only the flight components bundle on app initialisation, we don't load every service bundle altogether. We could have also loaded train booking bundle but user at this moment does not want to see train service. When user wants to book a train and lands on the train section that time only we load the train components bundle. Bottom line is rather than bundling everything up in one go, developers bundle components based on use cases.

* In a small-scale application, it does not make sense to split bundle into chunks because one bundle size will be enough for the application and this won't cause any performance issue.

Where do we do on demand Loading?

We do on demand loading of a component in a place where bundle will get loaded on demand.

Let us create a component Instamart and load this component on demand inside app.js.

Instamart.js

src > components > Instamart.js > Instamart

```
1
2   const Instamart = () => {
3     return (
4       <div>
5         <h1>Instamart</h1>
6         <h1>100 of components inside of it.</h1>
7       </div>
8     )
9   }
10
11  export default Instamart
```

Instamart.js App.js

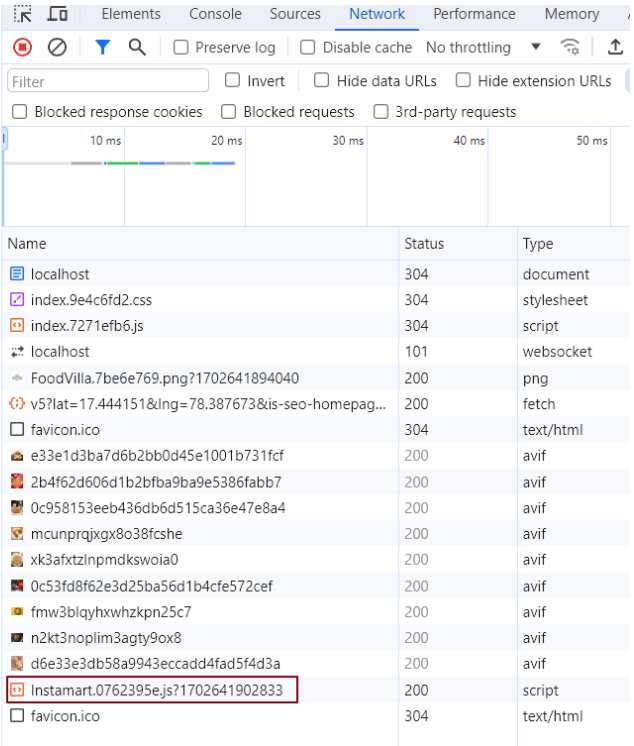
src > App.js > ...

```javascript
import React, { lazy } from "react";
import ReactDOM from "react-dom/client";
import Header from "./components/Header";
import Body from "./components/Body";
import Footer from "./components/Footer";
import About from "./components/About.js";
import { createBrowserRouter, RouterProvider, Outlet } from "react-router-dom";
import Error from "./components/Error";
import Contact from "./components/Contact";
import RestaurantMenu from "./components/RestaurantMenu";


// import Instamart from "./components/Instamart"; // Normal Import

const Instamart = lazy(() => import("./components/Instamart")); // Dynamic Import

const AppLayout = () => {
  return (
    <>
      <Header />
      <Outlet />
      <Footer />
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {…
      },
      {…
      },
      {…
      },
      {…
      },
      {
        path: "/instamart",
        element: (
          <>
            <Instamart />
          </>
        ),
      },
    ],
  },
]);

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<RouterProvider router={appRouter} />);
```

Food Villa	HomeAboutContact Cart Instamart

When we click on Instamart we load the bundle of Instamart component. This is on demand loading

undefined:undefined



Why undefined:undefined?

When we load instamart component, instamart component bundle takes some time to be loaded. Meanwhile react tries to render instamart component which does not exist, meaning this component is inside the bundler and bundler was not loaded. In this process react suspends this rendering. That's why we are getting the error component page in UI. At this moment err.Status and err.statusText will have undefined stored in it.

When we are loading our component on demand, react tries to suspend it.

Instamart card component is a suspense. We don't know whether it will be loaded on UI or not. It will load only when the bundle for this component is loaded.

App.js

src > App.js > AppLayout

```javascript
import React, { lazy, Suspense } from "react";
import ReactDOM from "react-dom/client";
import Header from "./components/Header";
import Body from "./components/Body";
import Footer from "./components/Footer";
import About from "./components/About.js";
import { createBrowserRouter, RouterProvider, Outlet } from "react-router-dom";
import Error from "./components/Error";
import Contact from "./components/Contact";
import RestaurantMenu from "./components/RestaurantMenu";
import ShimmerUI from "./components/ShimmerUI";

// import Instamart from "./components/Instamart"; // Normal Import

const Instamart = lazy(() => import("./components/Instamart")); // Dynamic Import

const AppLayout = () => {
  return (
    <>
      <Header />
      <Outlet />
      <Footer />
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {…
      },
      {…
      },
      {…
      },
      {…
      },
      {
        path: "/instamart",
        element: (
          <Suspense fallback={<ShimmerUI/>}>
            <Instamart />
          </Suspense>
        ),
      },
    ],
  },
]);

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<RouterProvider router={appRouter} />);
```

What is suspend?

<suspense> suspends the lazy loaded components until bundler for the component is available in UI

```
{
  path: "/instamart",
  element: (
    <Suspense fallback={<ShimmerUI/>}>
      <Instamart />
    </Suspense>
  ),
},
```

Now when we navigate to Instamart page, we see the contents of it because this time contents are coming from the Instamart component from loaded bundle.

Now React knows Instamart Component will be lazy loaded .so it waits for the promise to get resolved.

```
const Instamart = lazy(() => import("./components/Instamart")); // Dynamic Import
```

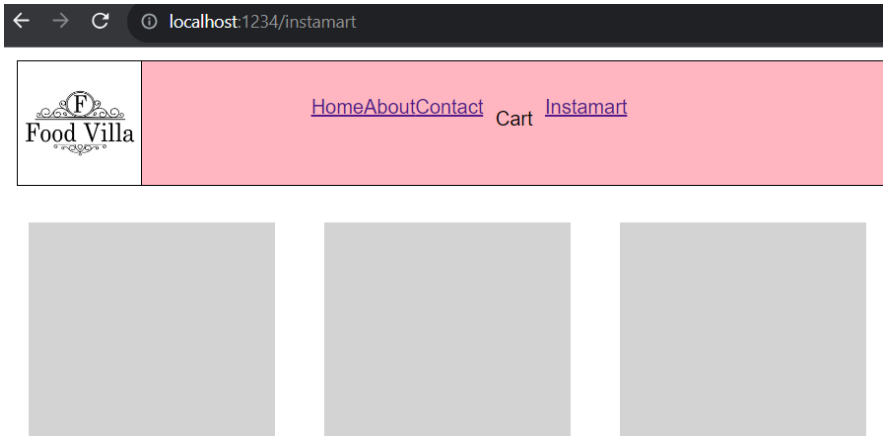
Import () returns a promise and react waits for the promise to get resolved when we wrap a component inside a suspense tag.

What is Fallback in react?

When bundle is not loaded react suspends the component for a while until the bundle is available. During this time for a better user experience we should display a shimmer UI on screen. So, suspense component accepts a prop called fallback where we call shimmer UI component. We can also write JSX in fallback attribute within {}.

```
{
  path: "/instamart",
  element: (
    <Suspense fallback={<ShimmerUI/>}>
      <Instamart />
    </Suspense>
  ),
},
```

Now if we navigate to the instamart page from home page.



When bundle is loaded-

Why Should you not Lazy load a component inside a component or parent component?

Because if we do, for ever state or prop change, parent component will be rendered. As a result, lazy loading of the child component takes place on every render. So, every time the bundle will be loaded.