Date..................

Chapter-09 Optimizing our App

CUSTOM HOOKS

→ why? when? and How?

→ why should we build hooks?
- * Reusability
- * Readability
- * Seperation of concerns
- * Maintainability

→ Hooks are just normal javascript functions.
→ functions are used to wrapup a logic and can reuse it anywhere we want to.

Eg:- In Body.js

filterData() is a function written seperately. We didn't write it between the code because it breaks modularity, reusability & readability.
If I need to use this function anywhere else then, I can reuse it.

● Great place to keep re-usable functions :-

→ Make a folder utils (It's utility, helper or common or whatever name).
→ Inside create a file utils and write the function filterData() inside it and make it named export.

Spiral Teacher's Sign ....

Date....................

→ Any no. of helperfunction can be kept inside utils.js

→ Advantage:
* functions become readable & reusable.
* This makes our code more testable because I can write seperate testcases for this.
* Maintainable because it is easy to debug.
* Modularity means we've broken down code into small pieces and every pieces have it's own responsibility.

Let us take <RestaurantMenu/> component. Job of this component is to show restaurant menu.
(i) find out restaurant id (resId).
(ii) get details of the restaurant from server.
(iii) to display.

we'll try to extract this logic. We will create a custom hook that will help us to get the restaurant details.

[NB: Create a hook with "use" name infront of it]

* Create a custom hook useRestaurant.
→ Create a new file named useRestaurant.js.

* code ⟶
└→ NEXT PAGE

Spiral   Teacher's Sign ....

Date.................

```
const useRestaurant = (resId) => {

  const [restaurant, setRestaurant] = useState(null);

  // Get data from(API)
  useEffect ( ()=> { getRestaurantInfo(); }, []);
  async function getRestaurantInfo() {
       const data = await fetch (
       "https://www.swiggy....." + resId);
     const json = await data.json();
     setRestaurant (json.data);
  }
  //return restaurantData
   return restaurant;

 };

export default useRestaurant;
```

Meanwhile, in <RestaurantMenu/>

```
 Const RestaurantMenu = () => {
       const {resId} = useParams();
       const restaurant = useRestaurant (resId);
       return !restaurant ? (<shimmer/>) :
         (
           .....
           .....); };

  export default RestaurantMenu;
```

Spiral   Teacher's Sign ....

Date..............

Next feature :⇒

Online & offline

① If the user have no internet connection, then it should show "You are offline, check your internet connection".
② Else it should show the data.

In Body.js →

```
const Body = () => {
    const isOnline = useOnline();
    if (!isOnline) {
        return (
            <h1> check your internet connection </h1>
        );
    };
```

Now we can create custom hook useOnline().

useOnline.js →

```
import { useState, useEffect} from "react";
const useOnline = () => {
    const [isOnline, setIsOnline] = useState(true);
    useEffect (() => {
        window.addEventListner ("online", () => {
            setIsOnline (true);
        });
        window.addEventListner ("offline, () => {
            set isOnline ( false);
        });
    }, []);
return isOnline;
}
export default useOnline;
```

function to check if user is online or offline

Teacher's Sign ....

Date.................

To fake offline

Go to chrome devtools → Network tab
→ change speed option (fast, slow 3G, offline)

Important (senior developer banna hai ?)

* cleaning cache

- whenever eventListner is added, we should clean it up.
- [whenever you are going offline & getting back online, eventListner is created. only once because we've empty dependency array].
- It is always a good practice to clear the eventListner when we go out of the component. Otherwise browser will keep hold those.
- to do that :- ↓

```
useEffect (()⇒ {
        const handleOnline = ()⇒ {              yahan k function ko
                set IsOnline (true);             naam de diya
        };                                       kyuki remove
                                                 eventListner
                                                 lagana hai.
        window.addEventListner ("online", handle Online);
    window.addEventListner ("offline", ()⇒ {} This function
                set IsOnline (false);
                                                 ↓
        });
                                        handle offline()

    return ()⇒ {
        window.removeEventListner ("online", handle Online);
        window.removeEventListner ("offline", handle offline);
    }
}, []);
return isOnline;
};
```

Spiral   Teacher's Sign..............

Date..................

For the whole code, parcel creates only one js file.
In this file, all the code is bundled together. So the size of this index.js file is large. But in production bundle size of this file should be small.

There would be a 100s of components in a large website like "makemytrip".
Suppose if all these are bundled together in a single index.js file, It will blast. It will make our app very slow.
So to build a large-scale production ready application we should do :-

"CHUNKING"

It is also called as :-
→ Code splitting
→ Dynamic Bundling
→ Lazy Loading

we cannot bundle everything in our app.
→ On Demand Loading
→ Dynamic Import

Making a new different bundle in our App

Let us create "Instamart"
→ create an Instamart component.
→ In App.js file, do chuncking :-

Teacher's Sign ....

Date.................

→ In App.js file, do chunking :-

App.js :

Instead of importing like this :-

⊛ import Instamart from "./components/Instamart";

Do lazy loading :

⊛ Const Instamart = lazy (()=> {

return import ("./components/Instamart")});

So, now the "index.js" file in dist folder won't have code of instamart. It is created as separate file while loading.

This is called ON-DEMAND LOADING.

"when you are loading your component in demand, react tries to suspend it".

So when instamart is loaded for the first time, we see an error message on screen. This is because, instamart file took 27 ms to get loaded. But react tries to render it before it get loaded. That's why error.

Solution for this

"Suspense" → we can wrap instamart inside suspense.

Date..............

App.js :- ↓

```
{ path : "/instamart",
   element : (
         <Suspense>
               <Instamart/>
         </suspense>
   );
};
```

React now knows that when there is a suspense, what will be loaded.

In the intermediate time, a shimmer should be shown.
So, there is a prop known as "fallback".
So, write :- ↓

```
<Suspense fallback={<shimmer/>}   >

</suspense>
```

NB :-
* Never ever dynamically load your component inside another component.

Spiral   Teacher's Sign ....