**Namaste React**

**Chapter 9: Optimizing our App!**

* **Lazy Loading**
* **Suspense**
* **Chunking**
* **How to create a custom hook**

When should we build our custom hook?

how should we build our custom hook?

why should we build our custom hook?

at the end of the day hooks are just normal javascript functions, and why we create a function? we can wrap a small logic inside a function and we can reuse it anywhere we want to. in other words, by doing this we have achieved modularity, reusability, readability.

readability- suppose there is a function named as onclick which will do filterdata on button click. so by reading the name of function we got some idea that on clicking the button some actions will happen as per the code.

reusability- if we want to use this onClick function in some other places we can use it without any problem.

while creating the hook we will create as a new file, and always create a hook by using a use name in front of it. it is a react way of knowing that it is a hook. thats why we see all the hook have the use word in it. e.g. usestate, useEffect etc.

**What if we do named export for hook?**

we always do named export if we have single file for multiple components. and if we have a single file for single component then no need of named export, we can do simple default export.

this will improve the identification of components or the functions of that file while sharing the code with multiple people.

after creating the hook file with name as useFILENAME, as we know hooks are just normal JS functions so we will write the function and we will export it. in this function we will have params to take the input data.

If normal function and hook are same why dont we use normal function only?

in normal functions we cannot use state variables or an other hooks like, useState, useEffect etc. but in case of of out custom hook we can use state variables and all other hooks.

How to deal with offline user or no internet on our web page?

to handle this we will use something known as Window: online event

const useOnline = () => {

const [isOnline, setIsOnline] = useState(true);

useEffect(() => {

window.addEventListener("online", () => {

setIsonline(true);

});

window.addEventListener("offline", () => {

setIsonline(false);

});

}, []);

return isOnline;

};

above code is very basic code for the operation, to improve the code we will do as follow.

whenever we add event listeners we need to clean up the event listeners. because what is happening is whenever we are going offline and coming online a new event listener is created. and when when we go out of our component that event listener will stay there.

It is always a good practice to clear all the event listeners that we create. otherwise our browser will hold the event listeners.

**how to remove the event listeners?**

in the return statement we will remove all the event listeners as below.

return () => {

window.removeEventListener("online", handleOnline);

window.removeEventListener("offline", handleOffline);

};

here we have extracted the logic of addEventListener in handleOnline and handleOffline respectively.

As we know that whatever files we have may be JS, html or css. when we take the production build our bundler will take all of them and make a single file of it.

e.g., all JS files will be combined to one index.js file and same will go for html and css files as well.

but in case of large-scale production applications if we do like this then that file will be gigantic and will get flooded with lots of code.

to avoid making this huge file we will do something known as chunking or code Splitting or Dynamic bundling or Lazy loading.

**Chunking/Code Splitting/Dynamic bundling/Lazy loading**

while bundling also we should not bundle everything in our app instead we should do logical bundling.

**How do we do bundle?**

**What is the logical bundling means actually?**

consider the example of makeMyTrip which is used for booking hotels, trains, flights etc. if we have bundled all these together then code for all the mode of transportation will get loaded into our browser which will end up with slower response of page.

so what we can make is bundle each type of mode in different bundle, by doing this if user clicked on flights then only flights related bundle will get loads in the browser and same will happen for other as well.

to import a specific component into our app depending on some conditions like clicking on some buttons we will use something known as lazy().

so instead of normal import we will import using lazy(), here lazy is for lazy loading it is like a dynamic import.

this lazy() comes from our react library as a named import

import { lazy } from "react";

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

as we see in the above code lazy() takes a function which will do the dynamic import, and this import is same as below,

import BookFlights from "./components/BookFlights";

and if we take the build after doing this kind of lazy loading all the lazy loaded components will have a different bundled js file than the main index.js file.

NOTE: whenever we are importing our components on demand (lazy load) then upon render react tries to suspend the loading.

To handle this suspension of render of on demand component we will use something known as suspense

**What is Suspense?**

a <Suspense> component that lets you “wait” for some code to load and declaratively specify a loading state (like a spinner) while we’re waiting.

we will just wrap our dynamic import or lazy loading components inside the <suspense/> component.

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

<Suspense>

<BookFlights />

</Suspense>;

in above example lazy(() => import("./components/BookFlights")); this is like an promise, thats why it wait for the data/component to load without throwing an error.

but what if our lazy loading component takes more time, in this case what we can do is show shimmer or some spinner etc.

to do that <Suspense/> component has one prop named as fallback.

<Suspense fallback={<Shimmer/>}>

<BookFlights />

</Suspense>;

NOTE: We should never dynamically load/Lazy load our components inside another component

**why? -->** because this will lazy load aftr every render cycle. and this is not good practive to do so.