React (Day-9)

Optimising our app

- Always design components such that each component has only single responsibility. (Single Responsibility Principle)
- Always make your project modular, i.e., into different modules for better maintainability and testability.
- Following SRP & modularity helps us in better maintainance of our code, easy testing (writing unit test cases for each component) and better reusability.

* A good way to achieve SRP & modularity is to create custom hooks and make the components having multiple responsibilities (like fetching data ① and rendering data ②) follow SRP.

* To create a custom hook,
  - create hooks in utils folder
  - start the hook name with use
    - Ex: useRestaurantMenu

    React identifies hooks by "use" in the name of function.

* It is recommended by react to start hooks with "use" and name components in Pascal case. Even if you don't follow, your app will still work, but if you have a linter in your project setup, it might throw errors. Nevertheless, it's always a good practice to follow the recommended practices by reacts.

→ 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 (build) this file will be small.

→ But when building a large scale application, it would have 100's or 1000's of components. Suppose if we bundle them all, together in a single .js (index.js) file, it will be super slow & the file will be very large.

→ So, to build a large-scale production ready application, we should do "CHUNKING".

It is also known as :-

* Code splitting
* Dynamic bundling
* Lazy loading
* On demand loading
* Dynamic import

→ Let's say our food ordering app not only provides food, but also groceries.

→ Now we will make this groceries section as whole new different section of our project, using code splitting.

→ Make a Grocery.js component in the components folder.

→ In App.js, do chunking:-

Instead of importing the regular way

✗ import Grocery from "./components/Grocery";

Do Lazy loading:-

✓ const Grocery = lazy(named function provided by react (() => import("./comp...")));

path to Grocery component

→ So now, the index.js file in the dist folder (which was bundled due to dev build) won't have code of Grocery.

→ It is created as separate file while loading Grocery related js file is only created when we access the Grocery route for first time. This is called ON-DEMAND LOADING

NOTE: When we are trying to load our component on demand, react tries to suspend it.

So when Grocery is loaded for first time, we see an error message on screen.

Because, the Grocery (js file) took some 20 to 30ms to load. But react is fast AF. So when react tries to render Grocery component, the Grocery file was still loading, that's why the error :C

Solution? → named component provided by 'react'

"Suspense"!

↳ We can wrap our Grocery component inside ...Suspense.

App.js:

```
{ path: '/grocery',
  element: (
    <Suspense>
      <Grocery />
    </Suspense>
  )
}
```

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

In the mean time (when grocery file is being loaded) react won't throw an error, but an empty-white blank screen will be shown till grocery file is loaded.

→ So, it's better to have a fallback in Suspense.

```
<Suspense fallback={<Shimmer />}>
    <Grocery />
</Suspense>
```

Note: Never ever dynamically load your component inside another component.