Ep09Part05

Usually, a large-scale applications has a lot of components. Usually, the performance of these components are not that good.

Usually a bundler like parcel, bundles all files and make it into one file.

Everything that is happening to our webpage happens through one js file and that is the bundler file that is created by the bundler in our case it is parcel.

In the case of big projects or like startup which have a lot of components probably more than 1000 components in their project. In that case, having just a one js file will increase the size of that file which is not good.

For our own project, the restaurant app, the js file is almost 2 mb now and our project is not that big as well.

In development build the size is 2MB but in production build it will compress the file to less than 2MB, but think in terms of large projects which has a lot of components. Even if they release it to production build the size will still be much more larger. We don’t want that.

So to make the size of this file smaller, we have to break the app down into smaller pieces.

To do that we have to make smaller bundles of all the files. This process is known as app chunking or code splitting or dynamic bundling, or lazy loading and a lot more names.

We want to bundle, but then we don’t want to create 1000 files for each bundle and we don’t want to also put all the 1000 files in a single bundler. Both of the solutions is not valid.

We have to do logical separation of our bundle. It means a bundle should have enough code for our website to have one feature.

So how do we do this chunking?

Lazy loading: When our app loads it will not load the code for the component that we are not rendering right now. But it will only load when we go to that specific link where the code resides and only then it will be able to load to the UI and render it. The another word for is on demand loading.

When doing lazy loading we don’t import it as usual like we do for components.

Syntax:

import Grocery from “./components/Grocery”;

This is just a test component to simulate how lazy loading works. The grocery component is just there for testing purposes and it only contains a heading for now which displays it’s a grocery page and it has a lot of child components.

To load the Grocery component only when needed we need to use the word lazy in react or javascript.

Lazy is a function given to us by React package. It comes as a named export.

import {lazy} from “react”;

Lazy loading takes a callback function.

Syntax:

const Grocery = lazy(() => import(“./components/Grocery”));

Keep in mind that the import statement inside the lazy loading callback function is not the same as the normal import that we do in the top of our usual application. This import tak es a path as a parameter. It’s a function.

Now that we have implemented the lazy loading feature. If you go the bundler file which is the index.js file in the console you will not see the grocery component there.

But if were to go the grocery component now using the link that we created, it will load a new file called Grocery.js. It has created a new file for us altogether.

it will although not load the component and throw out an error in the console saying “React Router caught the following error during render Error: A component suspended while responding to synchronous input.”

The reason why the grocery component will throw this error because it takes a little of bit of time to load the component, but react try to render it quickly and see that there is no code inside the component so it decides to throw out an error saying that the component suspended while responding to synchronous input.

In other words, the page is not loaded yet, but react-router says let’s load this page. And these all things are happening during a click event so react says “I can’t just hang here, show me a fallback instead!.

To fix this we need to use the component SUSPENSE which is provided to us by react library.

We have to wrap our component in suspense

Syntax:

<Suspense fallback={<h1>Loading…</h1>}><Grocery/></Suspense>

We need to include a placeholder inside the suspense. It’s a prop to suspense called fallback. We can pass any component or any JSX in this fallback prop.

So, while the code is being loaded it will be shown as Loading…