React Code Splitting

The React app bundled their files using tools like **Webpack** or **Browserfy**. Bundling is a process which takes multiple files and merges them into a single file, which is called a **bundle**. The bundle is responsible for loading an entire app at once on the webpage. We can understand it from the below example.

**React 16.6.0**, released in **October 2018**, and introduced a way of performing code splitting. Code-Splitting is a feature supported by Webpack and Browserify, which can create multiple bundles that can be dynamically loaded at runtime.

A section of the app (or route) might import a large number of components that are not necessary when it first loads. This hurts the initial load time of our app.

You might have noticed that Create React App will generate one large .js file while we are building our app. This contains all the JavaScript our app needs. But if a user is simply loading the login page to sign in; it doesn’t make sense that we load the rest of the app with it. This isn’t a concern early on when our app is quite small but it becomes an issue down the road. To address this, Create React App has a very simple built-in way to split up our code. This feature unsurprisingly, is called Code Splitting.

**Advantages:**Though you have not reduced code writing you have reduced the code needed to be downloaded. This lazy load can improve the performance of your web application dramatically.

Code splitting uses **React.lazy** and **Suspense** tool/library, which helps you to load a dependency lazily and only load it when needed by the user.

The code splitting improves:

* The performance of the app
* The impact on memory
* The downloaded Kilobytes (or Megabytes) size

### React.lazy

The best way for code splitting into the app is through the dynamic **import()** syntax. The React.lazy function allows us to render a dynamic import as a regular component.

const ExampleComponent = React.lazy(() => import('./ExampleComponent'));

function MyComponent() {

return (

<div>

<ExampleComponent />

</div>

);

}

### Suspense

If the module which contains the ExampleComponent is not yet loaded by the function component(MyComponent), then we need to show some **fallback** content while we are waiting for it to load. We can do this using the suspense component. In other words, the suspense component is responsible for handling the output when the lazy component is fetched and rendered.

const ExampleComponent = React.lazy(() => import('./ ExampleComponent'));

function MyComponent() {

return (

<div>

<Suspense fallback={<div>Loading...</div>}>

<ExampleComponent />

</Suspense>

</div>

);

}

The **fallback** prop accepts the React elements which you want to render while waiting for the component to load. We can combine multiple lazy components with a single Suspense component.

const ExampleComponent = React.lazy(() => import('./ ExampleComponent'));

const ExamComponent = React.lazy(() => import('./ ExamComponent'));

function MyComponent() {

return (

<div>

<Suspense fallback={<div>Loading...</div>}>

<section>

<ExampleComponent />

<ExamComponent />

</section>

</Suspense>

</div>

);

}

Ex: ======================

import React from 'react'

export default () => {

return(

<div>

<hr/> <h1> Hi MyComp </h1>

</div>

);

}

import React,{Component,lazy, Suspense} from 'react'

const MyComp = lazy( () => import('./MyComp'));

class App extends Component {

render() {

return (

<div>

<Suspense fallback={<div> <h1>Loading...</h1></div>}>

<div>

</div>

<MyComp/>

</Suspense>

</div>

);

}

}

export default App