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.

**App.js**

1. **import** { add } from './math.js';
3. console.log(add(16, 26)); // 42

**math.js**

1. export function add(a, b) {
2. **return** a + b;
3. }

**Bundle file as like below:**

1. function add(a, b) {
2. **return** a + b;
3. }
5. console.log(add(16, 26)); // 42

As our app grows, our bundle will grow too, especially when we are using large third-party libraries. If the bundle size gets large, it takes a long time to load on a webpage. For avoiding the large bundling, it?s good to start ?splitting? your bundle.

**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.

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.

**Before**

1. **import** ExampleComponent from './ExampleComponent';
3. function MyComponent() {
4. **return** (
5. <div>
6. <ExampleComponent />
7. </div>
8. );
9. }

**After**

1. **const** ExampleComponent = React.lazy(() => **import**('./ExampleComponent'));
3. function MyComponent() {
4. **return** (
5. <div>
6. <ExampleComponent />
7. </div>
8. );
9. }

The above code snippet automatically loads the bundle which contains the ExampleComponent when the ExampleComponent gets rendered.

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.

1. **const** ExampleComponent = React.lazy(() => **import**('./ ExampleComponent'));
3. function MyComponent() {
4. **return** (
5. <div>
6. <Suspense fallback={<div>Loading...</div>}>
7. <ExampleComponent />
8. </Suspense>
9. </div>
10. );
11. }

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. It can be seen in the below example.

1. **const** ExampleComponent = React.lazy(() => **import**('./ ExampleComponent'));
2. **const** ExamComponent = React.lazy(() => **import**('./ ExamComponent'));
4. function MyComponent() {
5. **return** (
6. <div>
7. <Suspense fallback={<div>Loading...</div>}>
8. <section>
9. <ExampleComponent />
10. <ExamComponent />
11. </section>
12. </Suspense>
13. </div>
14. );
15. }

**Note:** React.lazy and Suspense components are not yet available for server-side rendering. For code-splitting in a server-rendered app, it is recommended to use Loadable Components.

Error boundaries

If any module fails to load, for example, due to network failure, we will get an error. We can handle these errors with Error Boundaries. Once we have created the Error Boundary, we can use it anywhere above our lazy components to display an error state.

1. **import** MyErrorBoundary from './MyErrorBoundary';
2. **const** ExampleComponent = React.lazy(() => **import**('./ ExampleComponent'));
3. **const** ExamComponent = React.lazy(() => **import**('./ ExamComponent'));
5. **const** MyComponent = () => (
6. <div>
7. <MyErrorBoundary>
8. <Suspense fallback={<div>Loading...</div>}>
9. <section>
10. <ExampleComponent />
11. <ExamComponent />
12. </section>
13. </Suspense>
14. </MyErrorBoundary>
15. </div>
16. );

Route-based code splitting

It is very tricky to decide where we introduce code splitting in the app. For this, we have to make sure that we choose the place which will split the bundles evenly without disrupting the user experience.

The route is the best place to start the code splitting. Route based code splitting is essential during the page transitions on the web, which takes some amount of time to load. Here is an example of how to setup route-based code splitting into the app using React Router with React.lazy.

1. **import** { Switch, BrowserRouter as Router, Route} from 'react-router-dom';
2. **import** React, { Suspense, lazy } from 'react';
4. **const** Home = lazy(() => **import**('./routes/Home'));
5. **const** About = lazy(() => **import**('./routes/About'));
6. **const** Contact = lazy(() => **import**('./routes/Contact'));
8. **const** App = () => (
9. <Router>
10. <Suspense fallback={<div>Loading...</div>}>
11. <Switch>
12. <Route exact path="/" component={Home}/>
13. <Route path="/about" component={About}/>
14. <Route path="/contact" component={Contact}/>
15. </Switch>
16. </Suspense>
17. </Router>
18. );

Named Export

Currently, React.lazy supports default exports only. If any module you want to import using named exports, you need to create an intermediate module that re-exports it as the default. We can understand it from the below example.

**ExampleComponents.js**

1. export **const** MyFirstComponent = /\* ... \*/;
2. export **const** MySecondComponent = /\* ... \*/;

**MyFirstComponent.js**

1. export { MyFirstComponent as **default** } from "./ExampleComponents.js";

**MyApp.js**

1. **import** React, { lazy } from 'react';
2. **const** MyFirstComponent = lazy(() => **import**("./MyFirstComponent.js"));