Lazy Loading React Components (with react.lazy and suspense)

Nwose Lotanna Follow
Jan 31, 2019 · 7 min read

Photo by Victoria Heath on Unsplash

Sometime last year, the team at React released the version 16.6.0 which shipped with a shiny new feature that would simplify the way we handle lazy loading without the help of third-party libraries.

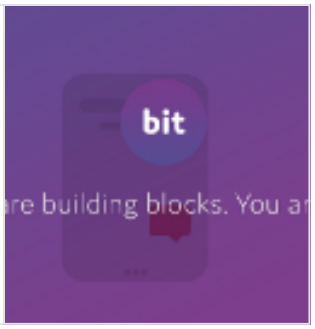
Let's take a look at how you can leverage this feature in your application in order to improve performance and build a better experience for users.

As always, when building with components it's useful to organize them in a collection using tools like **Bit**. Then you can share and use your components in any app you'd like, to speed development and keep your code DRY. Here it is:

Component Discovery and Collaboration · Bit

Bit is where developers share components and collaborate to build amazing software together. Discover components shared...

bit.dev



What is React.lazy()

It is a new function in react that lets you load react components lazily through code splitting without help from any additional libraries. Lazy loading is the technique of rendering only-needed or critical user interface items first, then quietly unrolling the non-critical items later. It is now fully integrated into core react library itself. We formerly used *react-loadable* to achieve this but now we have *react.lazy()* in react core.

Suspense

Suspense is a component required by the lazy function basically used to wrap lazy components. Multiple lazy components can be wrapped with the suspense component. It takes a fallback property that accepts the react elements you want to render as the lazy component is being loaded.

Why is Lazy Loading (& Suspense) Important

Firstly, bundling involves aligning our code components in progression and putting them in one javascript chunk that it passes to the browser; but as our application grows, we notice that bundle gets very cumbersome in size. This can quickly make using your application very hard and especially slow. With Code splitting, the bundle can be split to smaller chunks where the most important chunk can be loaded first and then every other secondary one lazily loaded.

Also, while building applications we know that as a best practise consideration should be made for users using mobile internet data and others with really slow internet connections. We the developers should always be able to control the user experience even during a suspense period when resources are being loaded to the DOM.

Getting Started

According to the react official documentation, you have webpack bundling already configured for use out of the box if you use:

- CRA (create react app)
- Next js
- Gatsby

If you are not, you will need to setup bundling yourself. For example, see the Installation and Getting Started guides on the Webpack official documentation.

Demo

We are going to build a react application that displays names and number of albums of headline artists for MTV Base in 2019 using the create-react-app starter tool and then implement lazy loading with suspense in it. I have cleaned up a create-react-app to something simpler and built a simple component which we will use in this tutorial.

- Clone the repository below:

Nwose Lotanna / react-lazy-load

GitLab.com

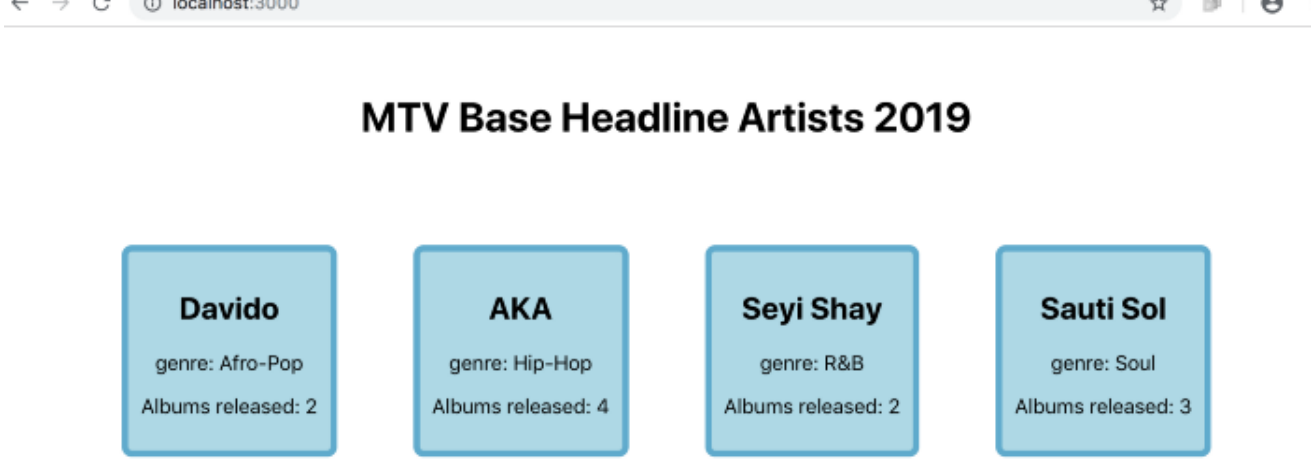
gitlab.com

- unzip the file and open a terminal.
- Install the project's node modules in the root directory of the unzipped file with this line of command:

```
$ sudo npm install
```

- Start the development server with this line of command:

```
$ sudo npm start
```



Sample Application

Here is our simple application, if you cloned the repository you will see that the artists data is loaded from a store inside the application.

OR you can create one yourself and make the changes below, the *src* folder of your application should look like this:

1. *Artists.js*

```
import React from ‘react’;

import ‘./App.css’;

import artists from “./store”;

export default function Artists(){

 return (

   <>

   <h1>MTV Base Headline Artists 2019</h1>

   {artists.map(artist =>(

   <div id=”card-body” key={artist.id}>

    <div className=”card”>

     <h2>{artist.name}</h2>

     <p>genre: {artist.genre}</p>

     <p>Albums released: {artist.albums}</p>
```

```
    </div>
   </div>
    ))}
   </>
);
}
```

2. *Store.js*

```
export default [
{
  id: “1”,
  name: “Davido”,
  country: “Nigeria”,
  genre: “Afro-Pop”,
  albums: “2”
},
{
  id: “2”,
  name: “AKA”,
  country: “South-Africa”,
  genre: “Hip-Hop”,
  albums: “4”
},
{
  id: “3”,
  name: “Seyi Shay”,
  country: “Nigeria”,
```

```
  genre: “R&B”,
  albums: “2”
},
{
  id: “4”,
  name: “Sauti Sol”,
  country: “Kenya”,
  genre: “Soul”,
  albums: “3”
}
];
```

3. *Index.js*

```
import React from ‘react’;
import ReactDOM from ‘react-dom’;
import ‘./index.css’;
import Artists from ‘./Artists’;
class App extends React.Component {
 render(){
  return(
   <div className=”App”>
    <Artists />
   </div>
   );
 }
}
ReactDOM.render(<App />, document.getElementById(‘root’));
```

4. *App.css*

```
.App {
 text-align: center;
}
h1 {
 padding: 30px;
}
#card-body {
 display: inline-flex;
 padding: 10px;
 margin: 30px 30px;
 border: 5px solid rgb(93, 171, 207);
 border-radius: 8px;
 background: lightblue;
}
```

Now let us see how to use react.lazy and suspense to handle lazy loading of the artists component.

- Head to the *index.js* file and import lazy and suspense from react like this:

```
import { Suspense, lazy } from 'react';
```

- To render a dynamic import as a regular component, the react documentation gives the react.lazy function syntax like so:

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

function MyComponent() {
  return (
    <div>
      <OtherComponent />
```

```
    </div>
  );
}
```

- Trying it out with our Artists component we would have something like this:

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

function MyComponent() {
  return (
    <div>
      <Artists />
    </div>
  );
}
```

If the module containing the Artists is not yet loaded by the time my App component renders, we must show some fallback content while we're waiting for it to load. This can be a loading indicator, brought in action by the suspense component. Below is the syntax for adding suspense component to react.lazy:

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

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

With our artists component, this becomes:

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

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <Artists />
      </Suspense>
    </div>
  );
}
```

Putting it all together, your index.js file should be like this:

```
import React, { lazy, Suspense } from ‘react’;

import ReactDOM from ‘react-dom’;

import ‘./index.css’;

// import Artists from ‘./Artists’;

const Artists = lazy(() => import(‘./Artists’))

class App extends React.Component {

 render(){

  return(

   <div className=”App”>

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

     <Artists />

    </Suspense>

   </div>

  );

 }

}

ReactDOM.render(<App />, document.getElementById(‘root’));
```

On your localhost it should be really fast and you might not be able to spot the changes. You can however create a timer helper or just simulate a slower network that would be able to show you exactly how the changes occur in milliseconds. This can be done by:

- opening the dev tools on your browser
- choosing the network tab
- clicking on the online tab at the far right to reveal other options (presets)
- choosing fast 3G

Dev Tools

Now you can refresh your browser and watch how lazy loading occurs..

Still Loading...

Multiple Lazy Components

Let us quickly add a small component that renders a header and see how the react.lazy function handles it with only one suspense component.

Create a *performers.js* file in your src folder and add the code below:

```
import React from ‘react’;
```

```
import ‘./App.css’;

export default function Performers(){

 return (

  <>

  <h2>These are the MTV Base Headline Artists...</h2>

  </>

 );

}
```

Then add the lazy component line in the *index.js* file and it should now look like this:

```
import React, { lazy, Suspense } from ‘react’;

import ReactDOM from ‘react-dom’;

import ‘./index.css’;

const Artists = lazy(() => import(‘./Artists’))

const Performers = lazy(() => import(‘./Performers’))

class App extends React.Component {

 render(){

  return(

   <div className=”App”>

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

     <Artists />

     <Performers />

    </Suspense>

   </div>

  );

 }

}
```

```
ReactDOM.render(<App />, document.getElementById('root'));
```

This should now show the two lazily loaded components show up at once after the placeholder element from the suspense has been rendered.

The two lazy components loaded within one suspense

This is quite unlike loadable which would have to render the loading element for each lazy component.

⚠️ Important Notice

`React.lazy` and Suspense is not yet available for server-side rendering. If you want to do code-splitting in a server-rendered app, Loadable Components is highly recommended. It has a nice guide for bundle splitting with server-side rendering.

Conclusion

We have seen how to get started using the lazy and suspense components provided by react to load components lazily. The above example is really basic compared to the numerous possibilities these new features bring. You can tell me down in the comments on how you have implemented lazy loading in your react project, happy coding!

. . .

learn more

Understanding Error Boundaries in React

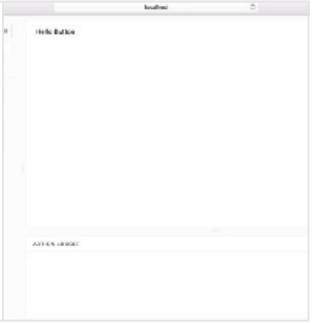
Learn how to use the new Error Boundaries in React to handles your Errors

blog.bitsrc.io

5 Tools for Faster Development in React

5 tools to speed the development of your React application, focusing on components.

blog.bitsrc.io



How To Write Better Code in React

9 Useful tips for writing better code in React: Learn about Linting, propTypes, PureComponent and more.

blog.bitsrc.io

React JavaScript Programming Web Development Software Engineering