# Code-Splitting

## Bundling

Most React apps will have their files "bundled" using tools like Webpack or Browserify. Bundling is the process of following imported files and merging them into a single file: a "bundle". This bundle can then be included on a webpage to load an entire app at once.

### Example

**App:**

```
// app.js
import { add } from './math.js';

console.log(add(16, 26)); // 42
```

```
// math.js
export function add(a, b) {
  return a + b;
}
```

**Bundle:**

```
function add(a, b) {
  return a + b;
}

console.log(add(16, 26)); // 42
```

**Note:**

> Your bundles will end up looking a lot different than this.

If you're using Create React App, Next.js, Gatsby, or a similar tool, you will have a Webpack setup out of the box to bundle your app.

If you aren't, you'll need to setup bundling yourself. For example, see the Installation and Getting Started guides on the Webpack docs.

## Code Splitting

Bundling is great, but as your app grows, your bundle will grow too. Especially if you are including large third-party libraries. You need to keep an eye on the code you are including in your bundle so that you don't accidentally make it so large that your app takes a long time to load.

To avoid winding up with a large bundle, it's good to get ahead of the problem and start "splitting" your bundle. Code-Splitting is a feature supported by bundlers like Webpack and Browserify (via factor-bundle) which can create multiple bundles that can be dynamically loaded at runtime.

Code-splitting your app can help you "lazy-load" just the things that are currently needed by the user, which can dramatically improve the performance of your app. While you haven't reduced the overall amount of code in your app, you've avoided loading code that the user may never need, and reduced the amount of code needed during the initial load.

## `import()`

The best way to introduce code-splitting into your app is through the dynamic `import()` syntax.

**Before:**

```
import { add } from './math';

console.log(add(16, 26));
```

**After:**

```
import("./math").then(math => {
  console.log(math.add(16, 26));
});
```

> **Note:**
> The dynamic `import()` syntax is a ECMAScript (JavaScript) proposal not currently part of the language standard. It is expected to be accepted in the near future.

When Webpack comes across this syntax, it automatically starts code-splitting your app. If you're using Create React App, this is already configured for you and you can start using it immediately. It's also supported out of the box in Next.js.

If you're setting up Webpack yourself, you'll probably want to read Webpack's guide on code splitting. Your Webpack config should look vaguely like this.

When using Babel, you'll need to make sure that Babel can parse the dynamic import syntax but is not transforming it. For that you will need babel-plugin-syntax-dynamic-import.

## Libraries

### React Loadable

React Loadable wraps dynamic imports in a nice, React-friendly API for introducing code splitting into your app at a given component.

**Before:**

```
import OtherComponent from './OtherComponent';

const MyComponent = () => (
  <OtherComponent/>
);
```

**After:**

```
import Loadable from 'react-loadable';

const LoadableOtherComponent = Loadable({
  loader: () => import('./OtherComponent'),
  loading: () => <div>Loading...</div>,
});

const MyComponent = () => (
  <LoadableOtherComponent/>
);
```

React Loadable helps you create loading states, error states, timeouts, preloading, and more. It can even help you server-side render an app with lots of code-splitting.

---

## Route-based code splitting

Deciding where in your app to introduce code splitting can be a bit tricky. You want to make sure you choose places that will split bundles evenly, but won't disrupt the user experience.

A good place to start is with routes. Most people on the web are used to page transitions taking some amount of time to load. You also tend to be re-rendering the entire page at once so your users are unlikely to be interacting with other elements on the page at the same time.

Here's an example of how to setup route-based code splitting into your app using libraries like React Router and React Loadable.

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Loadable from 'react-loadable';

const Loading = () => <div>Loading...</div>;

const Home = Loadable({
  loader: () => import('./routes/Home'),
  loading: Loading,
});

const About = Loadable({
  loader: () => import('./routes/About'),
  loading: Loading,
});

const App = () => (
  <Router>
    <Switch>
      <Route exact path="/" component={Home}/>
      <Route path="/about" component={About}/>
    </Switch>
  </Router>
);
```

**COMMUNITY**

Community Resources

Tools

**MORE**

Tutorial

Blog

Acknowledgements

React Native