# **Code-Splitting (Lazy Loading)**

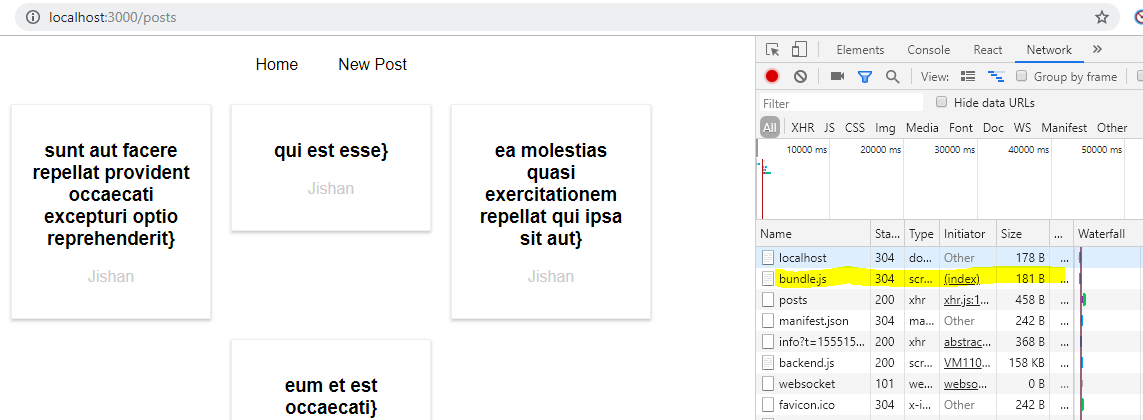
**Loading Routes Lazily (Loading Component Asynchronously)**

there's one more **advanced concept** though not difficult to implement but one advanced

concept I want to cover.

Let's go back to our application and to really show what I mean, let's go to the network tab in the developer tools. There if we have a look at all the requests once we load the page,

so if I go into posts you see that we're loading this bundle.js file



this contains all our source code, it's relatively big because we're in development mode, it will be much smaller once you are shipping this for production.

**Still loading the entire bundle with all the code of our application up front can be bad if we have a big application with distinct features and distinct areas in the app where a user might never visit a certain area. Like in our app, we have the NewPost.**

**if the user never visits new post, loading the code responsible for that component doesn't make a lot of sense.  It should only be loaded if the user actually navigates to /newposts ,**

**otherwise new posts and all potential children are never needed.**

**So why should we download the code up front? Would it be better to not download it and hence have a smaller upfront chunk to download and instead download the code responsible for this component and its children when needed.**

Now for a tiny application as ours here, this is not super useful because making that extra request for one kilobyte or something, what this component is worth when it comes to its size, that's not super useful but it is useful to know this technique for real for bigger applications you are building where this might matter where you are downloading quite a bit.

**The technique of downloading only what you need is known as code splitting or lazy loading and there you would essentially want to make sure that in your component, you're only loading the component once you need it.**

How does this work?

**To implement code splitting or lazy loading, we have create react app and react route for and that's important. This technique will work for react router for and for create react app**

**because code splitting depends heavily on the webpack configuration you are using, it is an advanced concept after all.**

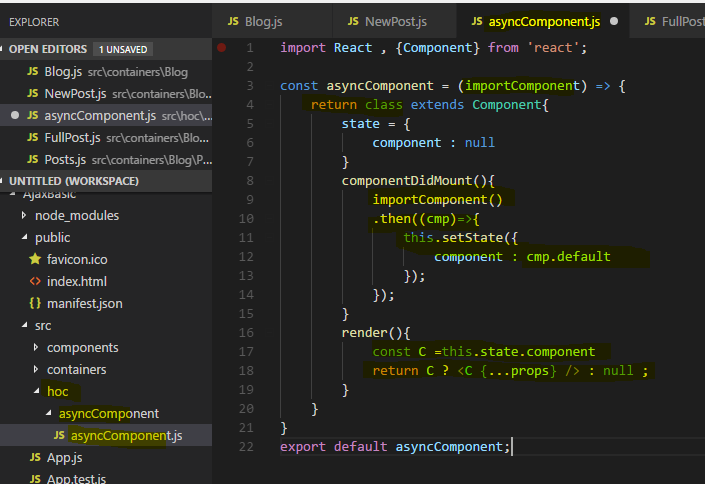
So the way I'm showing you is the way it works with the config from create react app which is a pretty modern and good configuration though, so chances are it also works in any decently set up webpack project

As I said at the beginning of this course, I strongly recommend using create react app anyway.

So **for this to work in this setup, we need a higher order component**.

So let's create a new folder hoc and then there, I'll add a new component which I'll name AsyncComponent

AsyncComponent.js , that's the javascript file name, because this component or this code here should help me load a component asynchronously i.e. only when it's needed.



**There I expect to get my import component argument which will in the end be a function**

So there, I now need to return something and I will return a class here which extends component, so a normal react component

I'll set up state and there I want to have a state which with a component property which is null. This state here, this component property will be set to the dynamically loaded component and the code for this will get implemented in componentDidMount.

So once this component was mounted here, this wrapping higher order component.

Now as **I said import component should be a function reference in the end, so what I want to do is I want to execute import component here and this actually will be a function**

**which will return as a promise , .then block will get an argument, cmp , the name is up to you which will have one property default which will be the component we loaded dynamically.**

**So in this .then block, I can call this.setState and set my component state to cmp.default. This is the case due to be set up we're using here with create react app.**

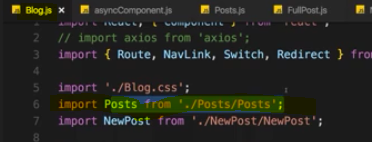
It is all of course heavily reliant on the type of function import component will refer to.

So now at some point of time, we will have loaded the actual component we want to use and it will be stored in our state. Hence in the render method, we want to render it, I'll create a constant and name it C and this should be this.state.component. Then I want to return jsx in this render method and I'll check if C is set in a ternary expression. If it is set, then I'll render C as a normal react component, I'll use this this.props spread trick here to pass any props we might need to this component and I'll set it to null if C is not set yet,

so if the component hasn't been resolved yet. Of course I now also need to export this asyncComponent function here.

Now we can go back to the blog component where we do import new posts,

**I want to load this dynamically now Now the thing is whenever you are importing something like this here, with import something from somewhere**



you basically inform webpack, the build tool which gets used behind the scenes about this dependency and it will include it in the global bundle, this is its job.

**Now for a lazy loading, this is exactly the opposite of what we want to do, we don't want to include it in the bundle, we want to load it when needed.**

**Still webpack needs to be able to dynamically prepare some extra bundle for this potentially loaded code.**

So what we have to do is we have to comment out this old way of importing

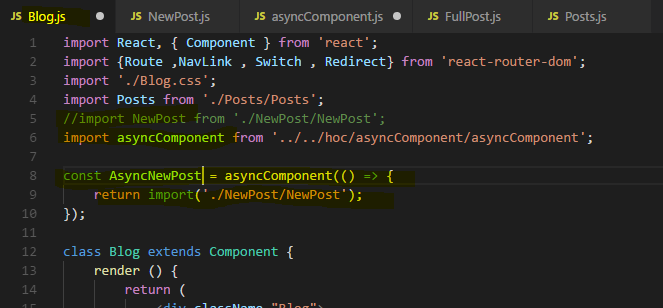
and I'll then use AsyncComponent here and execute it.

**Now asyncComponent this function, requires an argument and I told you that this argument, in that function we named it importComponent, that this argument should be a function which is why we executed like one here in componentDidMount**

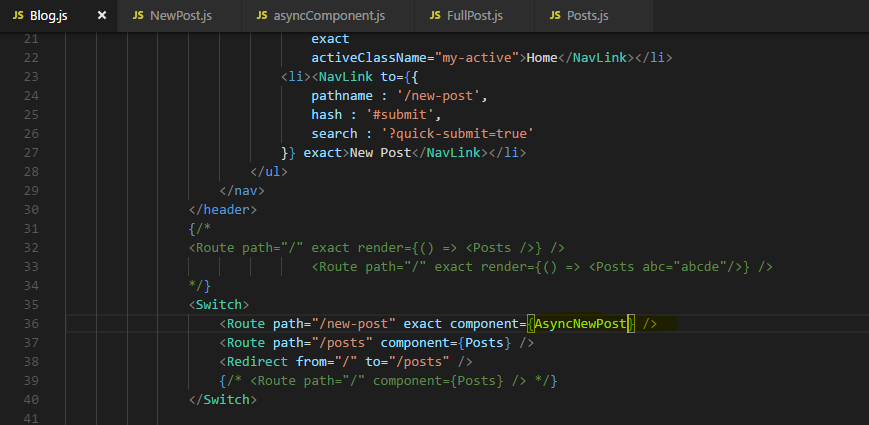
**So we have to pass some function to asyncComponent and this should be an anonymous function, there I'll use the import keyword as a function.**

**This is a special syntax, the dynamic import syntax which means whatever comes between the parentheses here is only imported when that function here is executed and that function here will only be executed once we render AsyncNewPost to the screen**.

now I'm only importing this when this constant gets used somewhere.



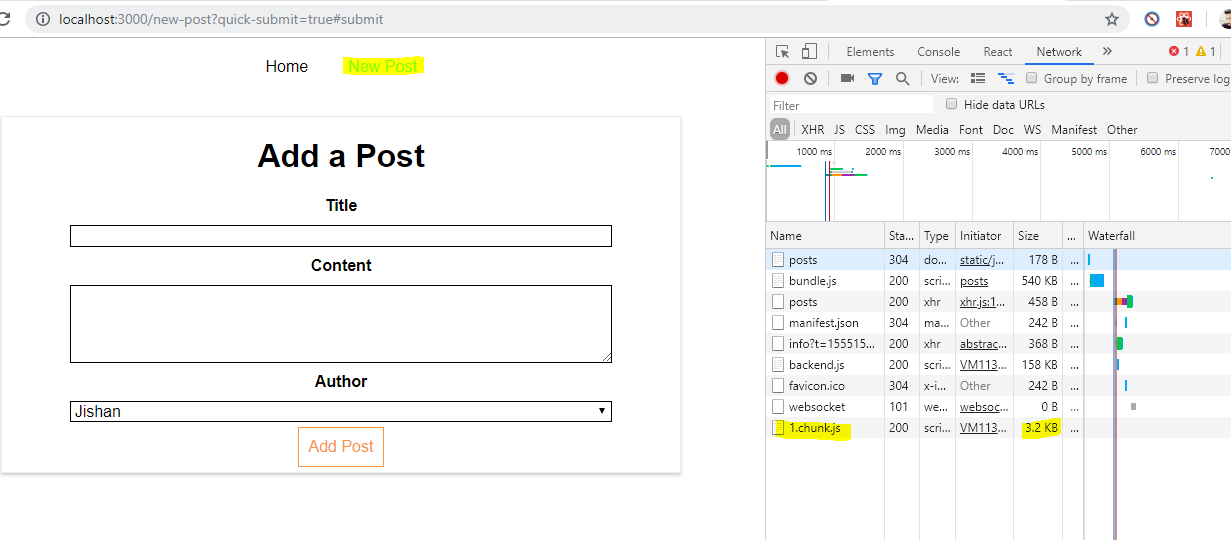
I want to use it down at the bottom of my blog container at the new post route, instead of using new post as a component, I want to use AsyncNewPost as a component.



**Eventually this will be a component because keep in mind AsyncComponent returns a component, we have a higher order component. It returns a class with a render method so this is a valid component.**

**This component will eventually render some dynamically loaded component and we decide which component it should be with the function we passed to AsyncComponent.**

**Now if we save this and we go back to our application, watch that path on the bottom right ,when I click on new post. Once I click there you'll see that this 1chunk.js file was loaded which is very small.**



This is an extra bundle webpack created because whilst bundling our code, it detected this dynamic syntax here which it knows due to our set up we're using too, this build workflow setup and therefore it created the extra bundle with the new post component and all potential child components that were exclusive to that component if any.

**But it didn't add it to the main bundle, instead it's prepared to load it when needed when we actually include AsyncNewPost which we only do when navigating to /newpost. This is how you load components asynchronously,**

and as I mentioned this is extremely useful in bigger apps where there are bigger chunks of code, a whole feature area in your application for example which might not even be visited by the user so you can save that code up front to only load it when needed.

## Lazy Loading with React Suspense(16.6 or above)

**if you are using the latest react version or react 16 or higher then you have a new way of lazy loading your routes because react 16.6 adds a new method on the react object.**

**The lazy method which you can use to load your data your components asynchronously. Which means only when they are needed.**

So react lazy is a method that was added with react 16.6.

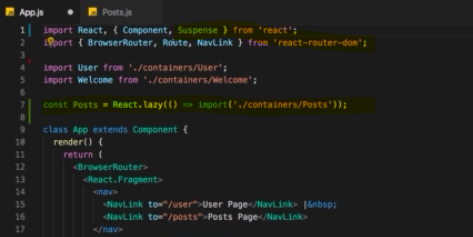
That allows us to load components asynchronously which means they are only loading, the code behind them is only loaded when they are really required when they are being rendered.

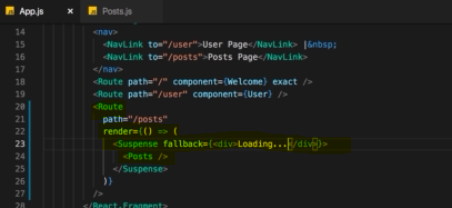
**And this is for example useful when having routing in your application because only when a user visits a certain route that component will be required and react lazy allows you to defer the rendering and the loading of the code of that component until it is required. And that of course means that you don't load redundant code in advance.**

It's not just useful for routing.

**Whenever you have a use case whereas some component is loaded at a later point of time for example because you have a check and some condition needs to be met to render a certain component in all such cases you could use react lazy.**

Lets see in Action-





**We should use default exports named exports are not supported in React.lazy() method.**

**there we now need to make a little change. Instead of using that component let's use the render method here.**

**I now need to import a new component from react Suspense component with an import.**

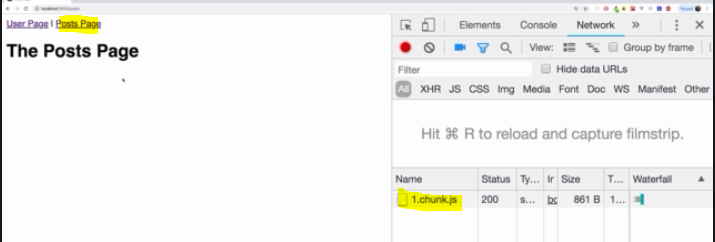
**That is what I want to render here.**

**And in-between all render posts.**

**So my constant as a component here and now let's add one more thing to that Suspense component. And that's the fallback props which should be JSX code and they all add div where I say loading and this will actually be displayed. In cases where re-act basically postpones the rendering of this wrap component and shows as well fallback in the meantime.**

**And of course that doesn't have to be a div with loading that could be a spinner or anything like that.**

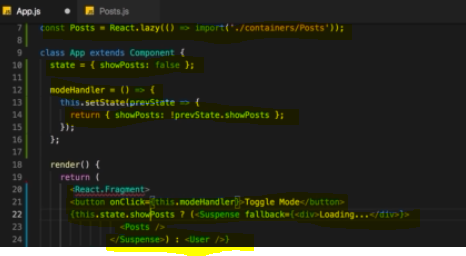
now click on the post page and you will see that there it loaded a new file and that is the file holding the code for this component and that is async rendering at async loading in action because this component and its code is only fetched and rendered when we really need it.

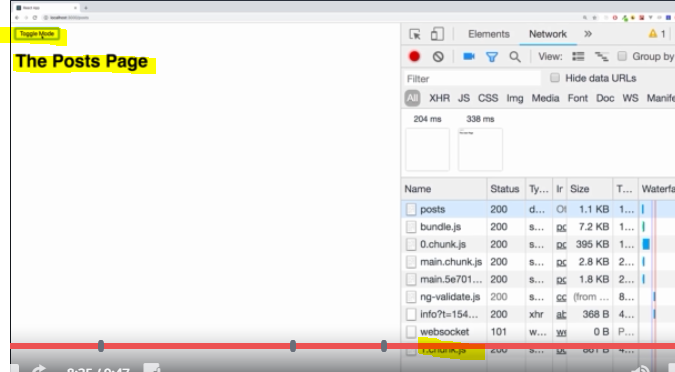


So this is a neat alternative to the other way of lazy loading pages which we had before.

A great advantage of this approach is that we of course can not just used is in a routing scenario.

We could of course also have a scenario where we don't use that browser router instead let's say I have a simple button here where I say toggle mode . And now I just want to render either the user or to post page.



if I go back and I click on toggle mode we see that chunk was loaded and we see the post page. 

So here we all take advantage of this. Async rendering and that is really something useful and a great addition with react 16.6

Please be aware that this will not work if you're trying server-side rendering.

Then this is not supported.

But if you have a client side rendered application as we have it here and of course then this is a great addition and definitely something you want to check out for cases where you conditionally render some component needed in the if statement or be dead in a routing scenario.

Now of course one thing to keep in mind the benefit you will get out of this will be greater if you have larger chunks of data behind your components.

If you have very simple components using suspends might actually be overkill and could even slow down your application or be suboptimal.

## React Official Doc

### Code-splitting

https://reactjs.org/docs/code-splitting.html

**bundling** - Most React apps will have their files “bundled” using tools like [Webpack](https://webpack.js.org/) or [Browserify](http://browserify.org/). 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.

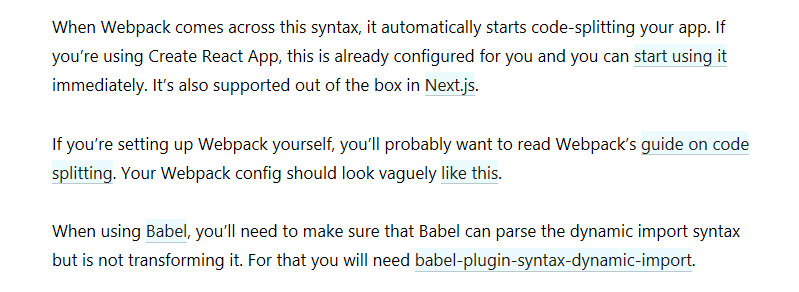
**Bundling is great, but as your app grows, your bundle will grow too. Especially if you are including large third-party libraries. You need to keep an eye on the code you are including in your bundle so that you don’t accidentally make it so large that your app takes a long time to load.**

**To avoid winding up with a large bundle, it’s good to get ahead of the problem and start “splitting” your bundle.** [Code-Splitting](https://webpack.js.org/guides/code-splitting/) is a feature supported by bundlers like Webpack and Browserify (via [factor-bundle](https://github.com/browserify/factor-bundle)) which can create multiple bundles that can be dynamically loaded at runtime.

**The technique of downloading only what you need is known as code splitting or lazy loading and there you would essentially want to make sure that in your component, you're only loading the component once you need it.**

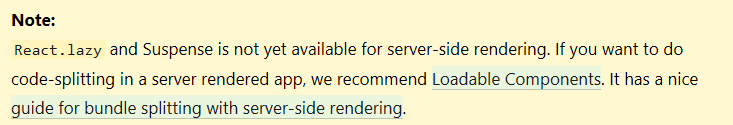
**Code-splitting your app can help you “lazy-load” just the things that are currently needed by the user, which can dramatically improve the performance of your app. While you haven’t reduced the overall amount of code in your app, you’ve avoided loading code that the user may never need, and reduced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





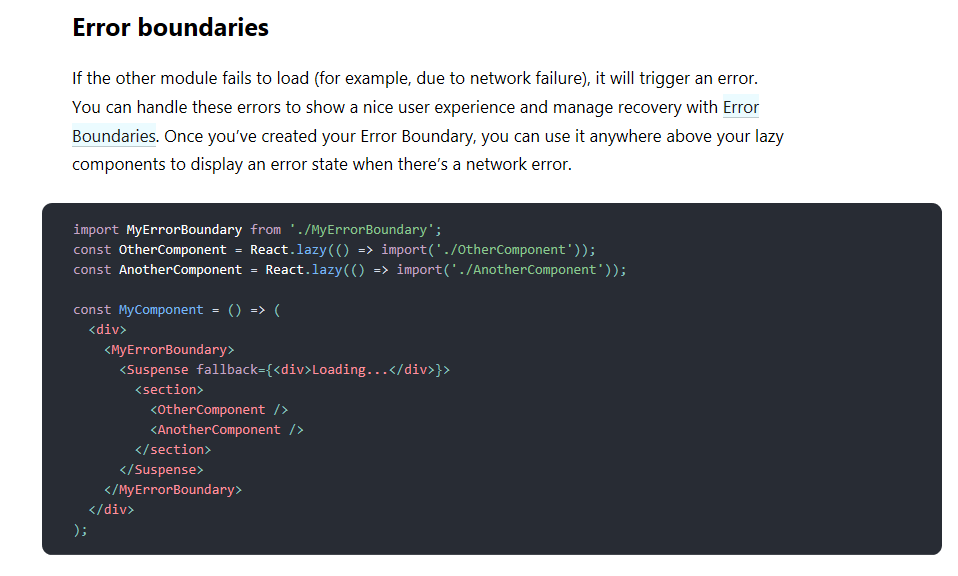
### **React.lazy – Lazy loading**

The React.lazy function lets you render a dynamic import as a regular component.





### Suspense



### Route-based code splitting