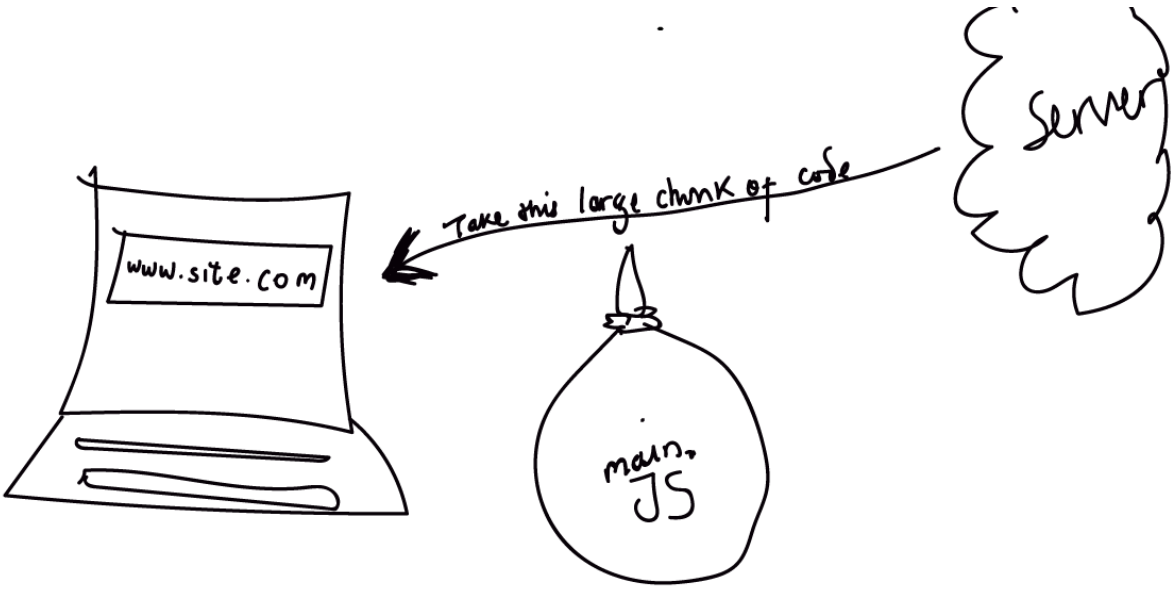
What is Lazy Loading?

Let's dive into the details of lazy loading in this lesson.

WE'LL COVER THE FOLLOWING

When you bundle your application, you will likely have the entire application bundled in one large chunk.

As your app grows, so does the bundle.



Example: Benny Game Application

To understand lazy loading, here's the specific use case Tunde had in mind when he discussed it with John.

"*Hey John, do you remember that the Benny app has an initial home screen?*".

By initial home screen, Tunde was referring to this:

This is the first screen the user encounters when they visit the Benny game. To begin playing the game, you must click the "Start Game" button to be redirected to the actual game scene.

"*John, the problem here is that we've bundled all our React components together and they are all served to the user on this page*".

"*Oh, I see where you're going*", said John.

"*Instead of loading the* `GameScene` *component and its associated assets, we could defer the loading of those until the user actually clicks 'Start Game', huh?*", said John.

And Tunde agreed with a resounding "*Yes, that's exactly what I mean*".

Lazy loading refers to deferring the loading of a particular resource until much later, usually when a user makes an interaction that demands the resource to be actually loaded. In some cases, it could also mean preloading a resource.

Essentially, the user doesn't get the lazy-loaded bundle served to them immediately, rather it is fetched much later at runtime.

immediately, rather it is fetched much later at runtime.

This is great for performance optimizations, initial load speed, etc.
React makes lazy loading possible by employing the dynamic import syntax.

Dynamic imports refer to a `tc39` syntax proposal for JavaScript. However, with transpilers like Babel, we can use this syntax today.

The typical, static way of importing a module looks like this:

```
import { myModule } from 'awesome-module'
```

Dynamically Loading Module

While this is desirable in many cases, the syntax doesn't allow loading a module dynamically at runtime.

Being able to dynamically load a part of a JavaScript application at runtime makes way for interesting use cases such as loading a resource based on a user's language (a factor that can only be determined at runtime), or only loading some code just when it is likely to be used by the user (performance gains).

For these reasons (and more), there's a proposal for introducing the dynamic import syntax to JavaScript.

Here's what the syntax looks like:

```
import('path-to-awesome-module')
```

It has a syntax similar to a function but is really not a function. It doesn't inherit from `Function.proptotype` and you can't invoke methods such as `call` and `apply`.

The returned result from the dynamic import call is a promise which is resolved with the imported module.

```
import('path-to-awesome-module')
        .then(module => {
    // do something with the module here e.g. module.default() to invoke the default export
 })
```

In the next lesson, we'll add React.lazy and Suspense to implement lazy loading.