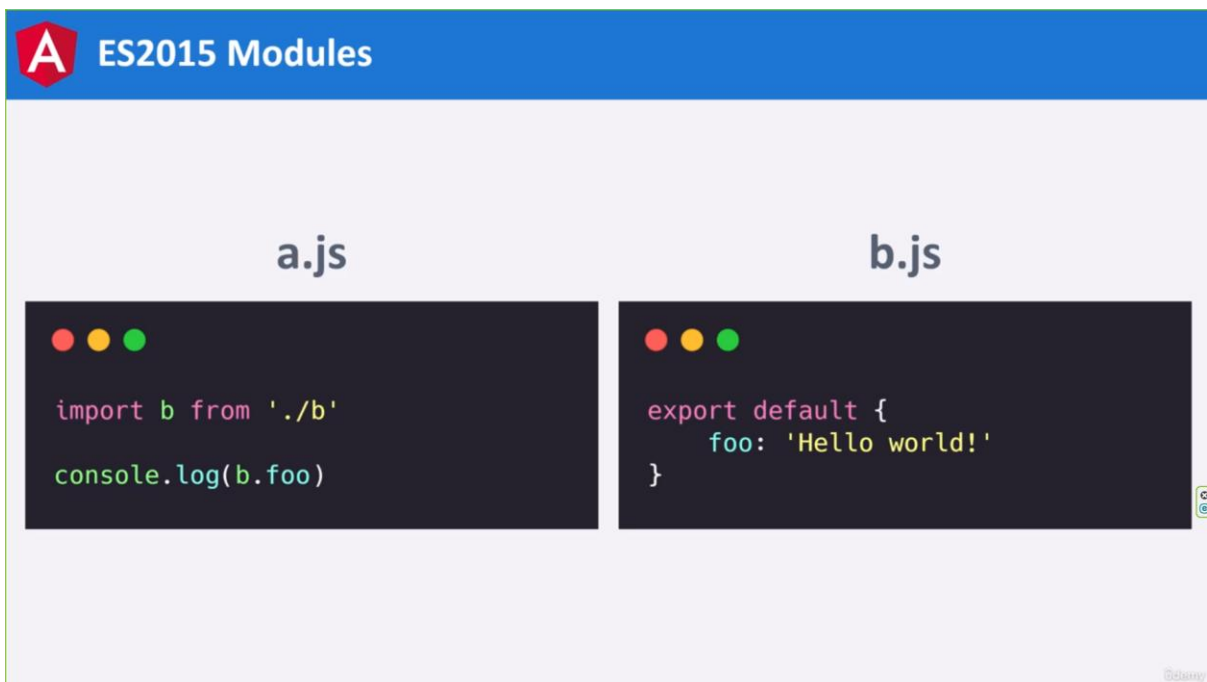


app.module – It's one of the most important files since it bundles our application.

Let's dive into what modules are first:



In JavaScript we have a module system, the goal of the system is to break code into separate files, this structure keeps our code maintainable, reusable and testable.

We can freely import and export code.


Here is an example of the module system in JavaScript.

We have a file called a.js, it imports the file called b.js.

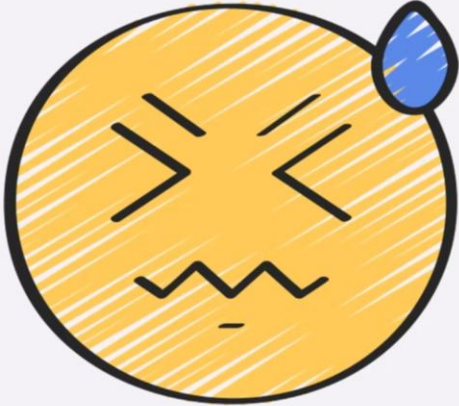
The b.js file is a module, it's exporting an object with a property called foo.

This is a very basic example of how modules work. They're very powerful and flexible. We have complete freedom over how we structure an application with modules.

Of course, life is never that easy, for small applications, this system works great, on the other hand, we can run into several problems with modules, just to name a few:


 **Problems with Modules**

- Duplicate Modules
- Circular Dependency
- Harder to manage overall




We can struggle with the duplicate modules and circular dependencies. Overall, it can be a struggle to keep track of everything.

The Angular team has designed an improved module system. You should think of it as a replacement for JavaScript modules. We will still need import in our app. Instead, you can think of it as an extension of the default module system. It's a system for helping us manage multiple dependencies by grouping them by feature:

 **Angular Modules**

- Not a replacement of ES2015 Modules
- Groups modules by feature



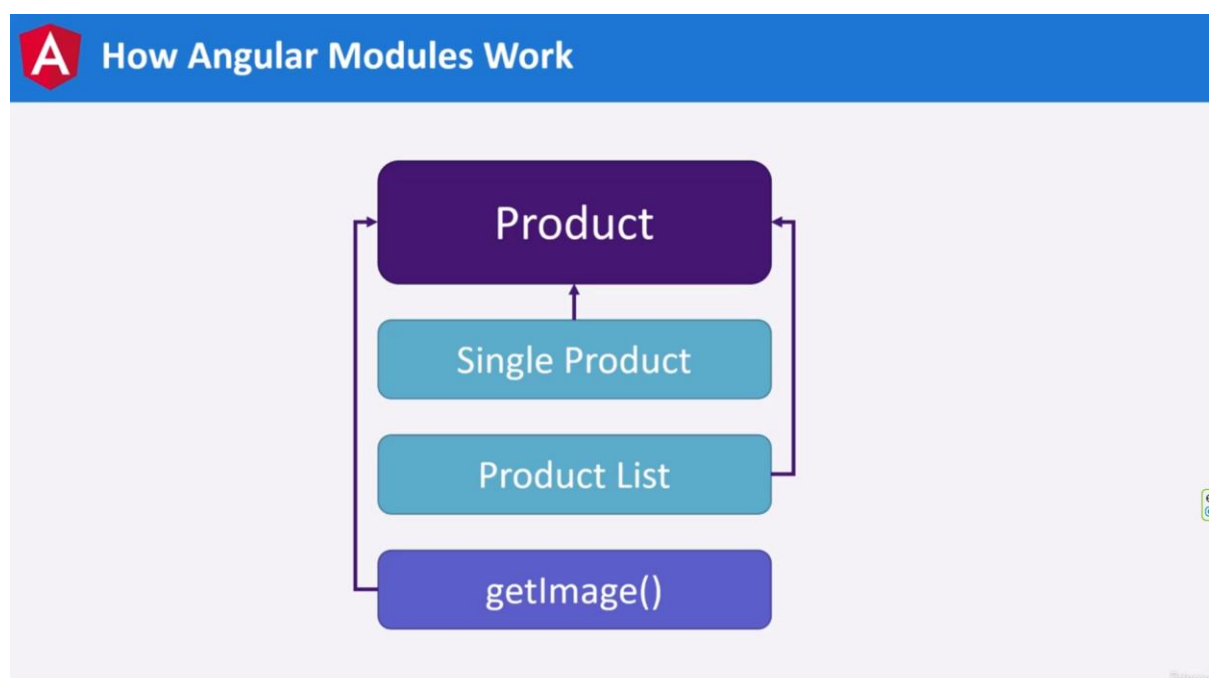
In a large app, we can have different sections, for example, let's imagine we are building an e-commerce application, we can have a product feature, this feature can comprise dozens of files, we may have a file for displaying a single product, another file for rendering a list of products and lastly, a function for retrieving an image for a product.

Each of these needs to be imported along with other files.

Instead of importing these files into each other, we can import them into an Angular module, by centralizing the imports into one file.



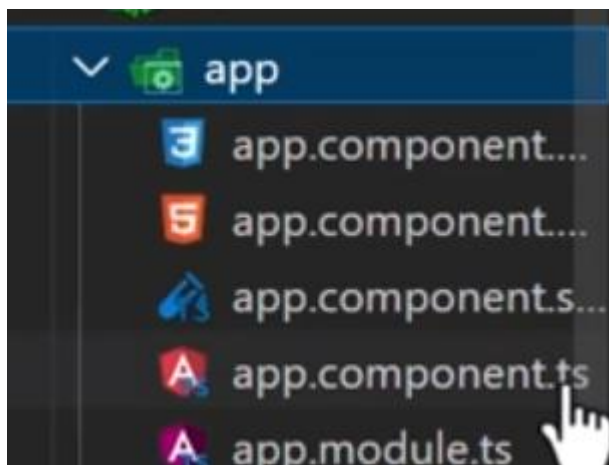
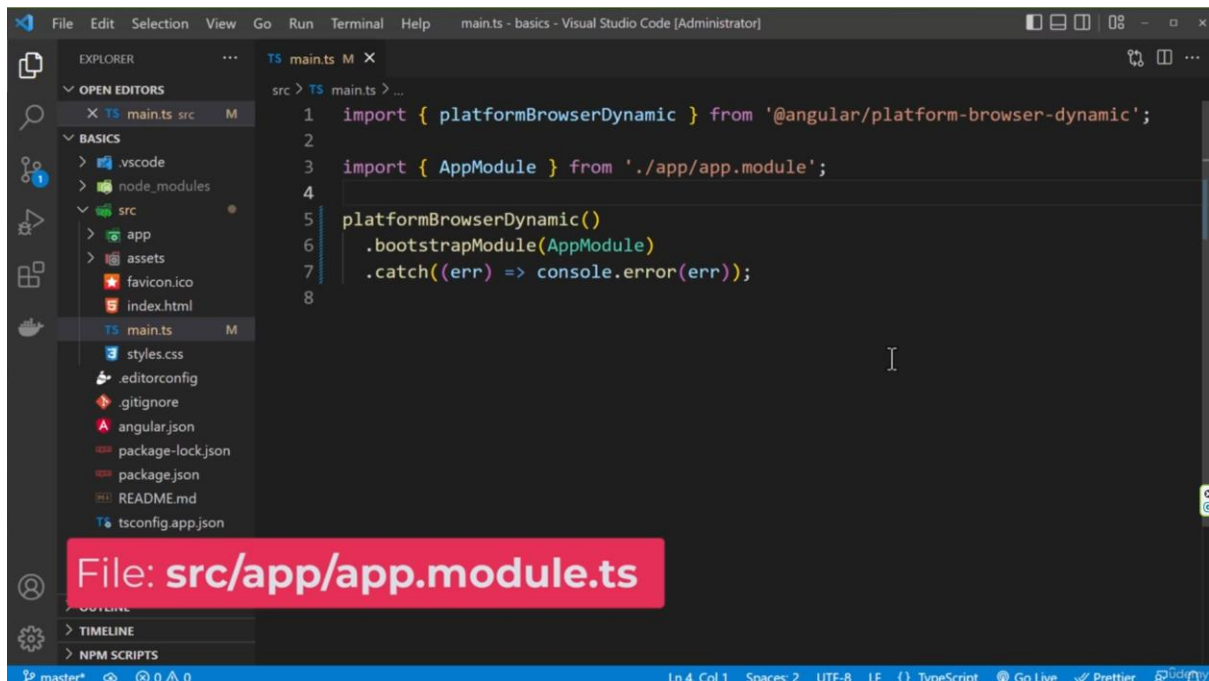
We can share code between files, the `getImage()` function can be called inside the single product file, even if it's not directly imported. Since they're both registered with the product module, the single product file has access to the `getImage()` function.



Angular excels at handling our modules.

Every application has a module called an app, we can consider this module the root module.

It can load other modules, which is something we'll be doing in a future lecture.



```

1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10  imports: [
11    BrowserModule
12  ],
13  providers: [],
14  bootstrap: [AppComponent]
15 })
16 export class AppModule { }
17

```

Rebuilding app.module.ts:

```

1 import { NgModule } from '@angular/core';
2 |
3 export class AppModule {
4
5 }

```

most be added

Decorator

```
src > app > app.module.ts > AppModule
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 @NgModule({
5   imports: [
6     BrowserModule
7   ]
8 })
9 export class AppModule {
10
11 }
```

Decorator