```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
})
export class AppModule { }
```

Since this is the root module, we need to tell Angular which component is the root component, i.e. the component that will be started when we bootstrap the app. That's what the `bootstrap` field of the configuration object is for:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Module ready, let's bootstrap the app!

8.6. Bootstrapping the app

Finally, we need to start our app, using the `bootstrapModule` method. This method is exposed on an object returned by a method called `platformBrowserDynamic`. You have to import it too, from `@angular/platform-browser-dynamic`. Now, that's a strange module! Why is it not `@angular/core`? Good question: it's because you might want to run your app somewhere else than in a browser, as Angular supports server-side rendering or running in a Web Worker for example. And in these cases, the bootstrap logic would be a bit different. But we'll see this later, as we are just focusing on the browser right now.