Entry components

An entry component is any component that Angular loads imperatively, (which means you're not referencing it in the template), by type. You specify an entry component by bootstrapping it in an NgModule, or including it in a routing definition.

> To contrast the two types of components, there are components which are included in the template, which are declarative. Additionally, there are components which you load imperatively; that is, entry components.

There are two main kinds of entry components:

- The bootstrapped root component.
- A component you specify in a route definition.

A bootstrapped entry component

The following is an example of specifying a bootstrapped component, `AppComponent`, in a basic `app.module.ts`:

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent] //
bootstrapped entry component
})
```

A bootstrapped component is an entry component that Angular loads into the DOM during the bootstrap process (application launch). Other entry components are loaded dynamically by other means, such as with the router.

Angular loads a root `AppComponent` dynamically because it's listed by type in `@NgModule.bootstrap`.

> A component can also be bootstrapped imperatively in the module's `ngDoBootstrap()` method. The `@NgModule.bootstrap` property tells the compiler that this is an entry component and it should generate code to bootstrap the application with this component.

A bootstrapped component is necessarily an entry component because bootstrapping is an imperative process, thus it needs to have an entry component.

A routed entry component

The second kind of entry component occurs in a route definition like this:

```
const routes: Routes = [
  {
    path: '',
    component: CustomerListComponent
  }
];
```

A route definition refers to a component by its type with `component: CustomerListComponent`.

All router components must be entry components. Because this would require you to add the component in two places (router and `entryComponents`) the Compiler is smart enough to recognize that this is a router definition and automatically add the router component into `entryComponents`.

The `entryComponents` array

Though the `@NgModule` decorator has an `entryComponents` array, most of the time you won't have to explicitly set any entry components because Angular adds components listed in

`@NgModule.bootstrap` and those in route definitions to entry components automatically. Though these two mechanisms account for most entry components, if your app happens to bootstrap or dynamically load a component by type imperatively, you must add it to `entryComponents` explicitly.

`entryComponents` and the compiler

For production apps you want to load the smallest code possible. The code should contain only the classes that you actually need and exclude components that are never used. For this reason, the Angular compiler only generates code for components which are reachable from the `entryComponents`; This means that adding more references to `@NgModule.declarations` does not imply that they will necessarily be included in the final bundle.

In fact, many libraries declare and export components you'll never use. For example, a material design library will export all components because it

doesn’t know which ones you will use. However, it is unlikely that you will use them all. For the ones you don't reference, the tree shaker drops these components from the final code package.

If a component isn't an *entry component* and isn't found in a template, the tree shaker will throw it away. So, it's best to add only the components that are truly entry components to help keep your app as trim as possible.

More on Angular modules

You may also be interested in the following:

- Types of NgModules
- Lazy Loading Modules with the Angular Router.
- Providers.
- NgModules FAQ.