Wednesday: Bootstrapping

Bootstrapping

All Angular apps have at least one Angular Module which is the root module that is used to launch the application. By convention, it is called the `AppModule` whose configuration is contained in the *app.module.ts* file.

Imports

app.module.ts

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


import { AppComponent } from './app.component';
import { GoalDetailsComponent } from './goal-details/goal-details.component';
import { GoalComponent } from './goal/goal.component';
import { StrikethroughDirective } from './strikethrough.directive';
import { DateCountPipe } from './date-count.pipe';
import { GoalFormComponent } from './goal-form/goal-form.component';

..........
```

Using the `AppModule` in our application, let's examine it. The first part is the Import section where we import external modules, components and directives that are used in our application.

NgModule decorator

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
    GoalDetailsComponent,
    GoalComponent,
    StrikethroughDirective,
    DateCountPipe,
    GoalFormComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

The `@ngModule` decorator identifies the `AppModule` as an `ngModule` class. It has metadata that tells Angular how to run the application.

Declarations array

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
    GoalDetailsComponent,
    GoalComponent,
    StrikethroughDirective,
    DateCountPipe,
    GoalFormComponent
  ],
 ......
```

Angular knows what apps belong to the `AppModule` by finding them in the declaration array. We also place custom directives and pipes that we create in the declarations array.
The Angular CLI's `generate` command adds the generated pipes, components and directives to the declaration array automatically for us.

Imports Array

app.module.ts

```
@NgModule({
.........
imports: [
    BrowserModule,
    FormsModule
  ],
```

In Angular, we group features into specific units called modules. We add a module to the imports array when the application wants to use its features. For example, since our application runs on the browser, we use features provided in the `BrowserModule`. We have also used some features like two-way data binding which are provided by the formsModule in our application

Providers Array

We can deliver services to different parts of an application using `dependency injection`. We use the providers array to register the different services needed by our application. We do not have any services yet that is why our array is empty. We'll learn about services later on and we'll see them being registered in this array.

Bootstrapping Array

app.module.ts

```
@NgModule({
.........
 bootstrap: [AppComponent]
 .......
```

We launch the application by bootstrapping the root component which is the `AppComponent`.

Bootstrapping an Application

main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

We bootstrap an Angular Application in the *main.ts* file. Here angular dynamically creates a browser platform and bootstraps the root Module which is the `AppModule`.The process of bootstrapping sets up an execution environment finding the `AppComponent` from the `bootstrap` array in the `AppModule` and creating an instance of it within its selector tag in the *index.html* file.

We have now seen why the AppModule is important in our application and where it is used.