Day 1 -- Your first angular project

Environment setup

Today you will learn how to setup a basic angular application from scratch. You will be using the following tools and concepts:

- npm
- package management
- angular cli
- angular components

Prerequisites

- Donwload and install NodeJS LTS version found under https://nodejs.org/en/download/.
- Open a command window and check that node was installed correctly by running:

```
> node -v
```

- Check that the npm package manager was installed correctly by running:

```
> npm -v
```

- Next intall the angular-cli tool.

```
> npm install -g @angular/cli
```

  This will install the angular command line tool which you will use for creating and managing all your future angular projects. To check that the CLI was intalled correctly, run the following command:

```
> ng help
```

- Last we advise you to pick a IDE for developping angular applications. Our personal favorite is **Visual Studio Code** found here https://code.visualstudio.com/Download. Go ahead and install it.

Creating a angular project

Now that you have your angualr development enviroment up and running, it is time to create your first project.

1. Create or choose a folder where you would like hold your angular application.

2. Open that folder inside VS Code.

3. Next, you will use VS Code's integrated command line for executing angular commands. Go ahead and run the following command in order to generate a angular project:

```
> ng new my-first-angular-project
```

4. You should now see the following folder structure in your project.

```
+---e2e/
|   +---app.e2e-spec.ts
|   +---app.po.ts
|   \---tsconfig.e2e.json
+---node_modules/
+---src/                              _
|   +---app/                           |
|   |   +---app.component.css          |
|   |   +---app.component.html         |
|   |   +---app.component.spec.ts      |-> source code of the application
|   |   +---app.component.ts           |
|   |   \---app.module.ts              |
|   +---assets/                       _|
|   |   \---.gitkeep
|   +---environments/
|   |   +---environment.prod.ts
|   |   \---environment.ts
|   +---favicon.ico
|   +---index.html <-- The html page that hold the angular application.
|   +---main.ts    <-- App entry point for the angular framework
|   +---polyfills.ts
|   +---styles.css <-- Golbal Style for the application
|   +---test.ts
|   +---tsconfig.app.json
|   +---tsconfig.spec.json
|   \---typings.d.ts
+---.angular-cli.json
+---.editorconfig
+---.gitignore
+---karma.conf.js
+---package-lock.json
+---package.json
+---protractor.conf.js
+---README.md
+---tree.txt
+---tsconfig.json
\---tslint.json
```

5. Start your angular app via this command and check the running application under http://localhost:4200 :

```
> npm start
```

Creating the Imitator component

Components are the main building block of angular applications, binding the views to the applications logic.

All angular components correspond to a certain **angular Module**. You will get more into angular modules later, for now, just keep in mind that no component can exista outside a certain module.

All components are usually composed from 3 files: `component.css component.ts and component.html`

If you look inside the generated project under `../src/app/` you will find that the Angular CLI generated the following 4 files:

```
app.component.html
app.component.css
app.component.ts
app.module.ts
```

The first 3 represent the `app.component`, which is the most important component of the application due to the fact that it hold all the application's content.

Because a component cannot live outside a module you can see that the `app.component` is referenced inside the `app.module`. Take a look. Don't worry if you don't understand all the code, just keep in mind that a component will be part of a module if it is declared inside that modules `declarations` array, like this:

```
@NgModule({
  declarations: [
    AppComponent // <-- AppComponent is part of this module
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent] // <-- `The AppComponent is set as the entry component
of the application !!! It has nothing to do with the bootrap CSS library`
})
export class AppModule { }
```

Like the `app.component`, the `app.module` is the most important module of a angular application, because it hold all other main modules of the application. You will have a better understanding of this once you will get to split your code into different modules. Also keep in mind that it is here where you can set the main entry components for your application via the `declaraions` array.

Now that you have your angular project structure all set and ready for running, it's time to write the code of your first angular component.

The component has the following specifications:

- it should be part of the `app.module`
- it should contain a text input field
- it should display whatever text is given to the input field at the start of the page.

Component implementation

Creating a component in angular is really easy when using the CLI. Create the **imitator** component by running the following command:

```
> ng generate component imitator
```

Your `app` folder structure should now look like this:

```
+---app/
|   +---imitator/
|   |   +---imitator.component.css
|   |   +---imitator.component.html
|   |   +---imitator.component.spec.ts
|   |   \---imitator.component.ts
|   +---app.component.css
|   +---app.component.html
|   +---app.component.spec.ts
|   +---app.component.ts
|   +---app.module.ts
|   \---tree.txt
\---tsconfig.app.json
```

The main file for for your component is `imitator.component.ts`. This file serves as a controller. Here you will implement all the logic of the component. Here you can also see how the component is linked to a coresponding html file and a css file.

Check the code bellow to understand what the code inside the imitator.component.ts file means.

```
import { Component, OnInit } from '@angular/core'; // Functions imported from the
Angular framework

@Component({ // Angular directive that transforms this class into a angular
component/

  // The component is used to inject the component inside a html page. We will get
to this, don't worry about it.
  selector: 'app-imitator-component',

  // Template url links the component logic to the component's html.
  templateUrl: './imitator-component.component.html',

  // Links the component to a stylesheet.
  styleUrls: ['./imitator-component.component.css']
})
export class ImitatorComponentComponent implements OnInit { // The component's
class declaration

  constructor() { } // The class constructor

  ngOnInit() { // Intialization hook function -- dont' worry about this for now
  }

}
```

The `imitator.component.html` file is a plain old html file in which serves as a **view** for our component. You will later see how to bind this view to the components controller.

We will not talk a lot about `imitator.component.css`. All you need to know is that this hold the style for the component view.

The `imitator.components.spec.ts` represents a test-bed environment for the imitator component. Don't worry about this for now.

You will see that the CLI allready declared the new component inside the `app.module`. Take a look.

Now, you will bind the component to it's corresponding view and implement the specifications.

1. First, open `imitator.component.ts` file and add a new property to the compoent class called `displayText`:

```
@Component({
selector: 'app-imitator-component',
templateUrl: './imitator-component.component.html',
styleUrls: ['./imitator-component.component.css']
})
export class ImitatorComponentComponent implements OnInit {

    displayText: string; //<-- add this

    constructor() { }

    ngOnInit() {
    }
}
```

2. Open the component's html file and replace it's content with the following code:

```
    <p>{{displayText}}</p>
    <input type="text" [(ngModel)]="displayText"/>
```

Don't worry if you don't understand the syntax for now.

Just keep in mind that double brackets like this `{{some_component_propery}}` serve as a place holder in the html. The brackets will be replaced with the value of the `displayText` property at runtime.

Also this piece of code `[(ngModel)]="displayText"` binds the value of the input field to our component's `displayText`.

The keyword `ngModel` is actually a nice angular component that attaches itself to the native html code and privides this functionality under the hood.

Because all components live insde a module, the `ngModel` lives inside the **FormsModule** which is a module provided by angular, and in order to use the `ngModel` compoenent you must first import it's host module.

To import the module add the following lines inside yout `app.module.ts` :

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


import { AppComponent } from './app.component';
import { ImitatorComponentComponent } from './imitator-component/imitator-component.component';


@NgModule({
declarations: [
    AppComponent,
    ImitatorComponentComponent
],
imports: [
    BrowserModule,
    FormsModule //<-- Add this
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
    ```
```

3. The component you just created is ready to use, so add it you your `app.component.html`; replace that file's content with the `selector` of your component:

```
<!--app.component.html -->
<app-imitator-component></app-imitator-component>
```

4. Start yout application via the `npm start` command or `ng serve --open`.

Congradulations on your first Angular component.
```