Monday: Angular Apps

Angular Apps

After that brief introduction to typescript and ES6, we can dive right into Angular. We will use the Angular *CLI* - **Command Line Interface** to generate a new Angular app.

```
$ ng new Goals
```

This command will take some time to run but in essence, it is creating a new angular application by providing us with boilerplate code and by installing the latest `npm` dependencies that are needed in our project.

When the command is done running, it creates a new Angular app with the files and folders that were generated while the command was running. We can view these files by opening the directory *Goals* in our favourite editor. It looks like this:

We can now go into our newly created project and run the app to see what we have. To run the app on the local development server, we use the *ng* serve command.

```
$ cd Goals
$ ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://local
host:4200/ **
```

This opens a live development server and so we can view our application on our web browsers. Let us put in the address `http://localhost:4200/ (http://localhost:4200/)` in our browser's URL tab. We can see the default Angular welcome page. But we haven't even written any code, right? So how is the browser displaying content?

Let us open the new Angular app in our favorite editor. We immediately see a lot of files that we did not create but was generated by the Angular CLI. We will ignore most of them for now and head directly to the `src/` folder. This is where we will write most of our code.

Every new Project is created with one **Component,** the `AppComponent`. We will discuss what Components later on. Inside our `src/` folder we have another subfolder called `app/` which is where our app component lives. We have a CSS file which holds the styles for the app-component and a HTML template file which holds the html code for the app-component. We also has a `app.comoponent.ts` file where we define all our logic for the app-component. Another file is the `app.module.ts` which stores all top-level configurations for the whole angular app. We'll take a look at it later.

In Angular, each component has a unique selector. This allows it to be called into any other component or to our `index.html` page.

src/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Goals</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

In our index page we render the `Appcomponent` component to our page using the `app-root` selector.

The `main.ts` file is responsible for importing the main module and bootstrapping the root component to be displayed first once we load the page.

Updating Our Components

We are going to be building a simple goal tracking application. We can start by updating our AppComponent.

app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h1> My Goals </h1>
    <p> My goal for today is {{goal}} </p>
  `
})
export class AppComponent {
  goal = 'Watch Finding Nemo';
}
```

We change the contents of our *app.component.ts* file. We create a property called `goal` and give it a value. We then use backticks to generate multiline HTML code. We use the string interpolation `{{}}` and the property to out template. Now we can serve the application and see the changes made.

Template File

Right now our application is very small and we can get away with writing our HTML code together with our logic. In larger apps, we need to separate our logic from the View file. But in a Larger application, this might be very confusing and difficult to maintain. Each component is created with a HTML template file we can use to render template code.

app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  goal = 'Watch Finding Nemo';
}
```

First, we remove the `template` parameter and replace it with `templateUrl` this will point to the template file.

app/app.component.html

```
<h1> My Goals </h1>
<p> My goal for today is {{goal}} </p>
```

We replace the initial content on the page with what was in our logic file. When we run the app again we should see nothing has changed.