Angular Tutorial (part 2: application)

Part II will demonstrate how to build a service class to retrieve data from the internet, and a component to display the data.

GETTING ORIENTED

Now that you've generated the trivial web application using the command line interface, it is time to build a basic app.

The ng new command generated over 200 megabytes of data with over 700 dependencies! These files provide Angular's functionality which includes built-in unit testing. To begin, find the index.html file.

Index.html will have a tag in the body called <app-root></app-root>. This tag refers to a component, called app, and will display any information contained in its component files.

If you want, paste a link to your favorite css framework inside the <head> tag at the top of the page. Now, to see the contents of <app-root>, navigate to the app folder inside the src folder.

MODIFYING <APP-ROOT>

The <app-root> component consists of four files: app.component.html, app.component.css, app.spec.ts and app.component.ts. The folder also contains an additional file called app.module which will be used for building a service class. app.spec.ts is for testing, so for now, just focus on the other three.

Open app.component.html, app.component.ts and app.component.css. First, in the .html file, delete everything below the </h1> tag. Next, navigate to the .ts file, where you will notice that a variable called "title" is defined.

Generally, variables defined in the ts file can be accessed within the html file by surrounding the variable with double brackets. This is called interpolation. Now, locate the title variable in the .ts file, and change it to something like "My First Angular App". Now, observe the {{title}} variable in the .html file. This calls the variable in the .ts file.

Finally, open app.component.css and modify the color of the h1 tag to something like:

```
h1{
        color: red;
}
```

To confirm your changes, save every file, and reload your browser at localhost:4200. If everything is loading, it is time to create a service class to retrieve data for your application.

BUILDING A SERVICE CLASS

It is best practice to generate the service class with the CLI. However, it is a good idea to keep ngserve running. So, open a second instance, and then type:

```
ng generate service api
```

This will create two new files in your app folder: api.service.ts and api.service.spec.ts.

To get the service class working, it will need to communicate with the internet. To do this, you need to import HttpClientModule and HttpClient into the into app.module and the service class. More information about these modules can be found at: https://angular.io/api/common/http/HttpClientModule and https://angular.io/guide/http.

To begin, open App.module.ts and paste the following to the top of the page.

```
import { HttpClientModule } from '@angular/common/http';
```

Next, include HttpclientModule in the imports array.

```
imports: [
  BrowserModule,
  HttpClientModule
]
```

Now, navigate to the newly generated service.ts file. This file will be used to communicate with the internet, so import the http client by adding the following statement to the top of the file.

```
import { HttpClient } from '@angular/common/http';
```

And then, make it accessible in the ApiService class by adding it to the constructor.

```
constructor (private http:HttpClient) {}
```

This creates an instance of an HttpClient called http which can be accessed by any ApiService object to communicate over the internet. The next step is to find some data to retrieve. To do this, you can use any API or database but for this tutorial, I'll use the free "countries" API to retrieve basic data points about every country in their database (more info can be found at https://restcountries.eu/).

Add the following function to the ApiService class (just under the constructor).

```
getCountries(){
 return this.http.get('https://restcountries.eu/rest/v2/name/united')     //all countries
}
```

Now, the service class is completed and ready be called on by your application.

MODIFYING <APP-ROOT> TO RECEIVE DATA

To use your new service class, it must be instantiated in one of the components. First, you'll do it in the <app-root> and then create an additional component to hold the data.

To begin, go to the app.component.ts file and import the service class, as well as the OnInit class, by pasting following statement over the existing import call.

```
import { Component, OnInit } from '@angular/core';
import {ApiService} from './api.service';
```

Now, add a constructor to the app component class as well as a variable called countryData (set to null). This will create an instance of the API service called apiService, and a variable, called countryData to display your data.

```
countryData = null;
constructor(private apiService:ApiService) {}
```

The next step is to add a function called ngInit which will call the getCountries function in the Service class.

```
ngOnInit() {
        this.api.getCountries().subscribe((data)=>{
        this.countryData = data;
});
```

As you can see, the function uses something called subscribe. This is called an observable and is like a promise. For more info see https://angular.io/guide/observables

MODIFYING <APP-ROOT> TO DISPLAY THE DATA

Now that the data has been brought into the component, it must be displayed. First, to make sure everything is working properly, you will do this with in <app-root>. Then, you will dynamically generate a new component to display the data.

To display the data in <app-root> you will need to iterate over the data retrieved from the service class. To do this, use *ngFor to loop over the objects in the countryData variable. Within the <h1> tag, paste the following code:

```
<p *ngFor= 'let country of countryData'>
  {{country.name}}
  Population: {{country.population}}
  Location: {{country.latlng}}
  </p>
```

This accesses the name, population and location of each country sent by the countries api. If you are using a different API, use your relevant keys. Just make sure countryData is the same variable name as the one you defined in your ts file.

Now, save all your files and check the application your browser at localhost:4200. It should display your data in the same color as the header. Now, it is time to create a new component to handle the data.

CREATING A NEW COMPONENT

Angular uses components to compartmentalize sections of code within an application. Components are accessed through tags and can be nested within each other. The command line interface is the best way to generate a component. To do this, make sure you are in your Angular folder and type:

```
ng generate component data
```

where data is the name of your component. This will register the new component with app.modules and generate four different files which are now in a folder called data within the app folder. Just like the app-root component, ignore the test file and open data.component.ts, data.component.html, and data.component.css.

First, observe the .ts file. In the decorator, @Component, and notice the line:

```
selector : ‘app-data’
```

This is the name of the tag that is used to access the contents of the component. It will be used when calling on the component in <app-root>. Now, to begin, you will need to add Input to the import statement at the top of the .ts file. It should look like:

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

This allows you to receive data from a parent component. Now, to specify the variable name, add the following code to the DataComponent class:

```
@Input() country                    // Capitalize Input!
```

This tells your component to look for a variable named country when it is called. Now it is time to nest the components. Go back to the app-root .html file, app.component.html. Copy all your data calls (within your <p> tags) and replace it with:

```
<app-data *ngFor= 'let country of countryData' app-data [country]="country">
</app-data>
```

This creates a new instance of your app-data component for every item in the object countryData. For each instance, it assigns the data object to the specified variable name. This functionality is made possible by the @Input declaration. If you used a different variable name, remember that the input variable in brackets “app-data [country]”, must correspond with the variable defined in the .ts file.

Now, navigate back to your data component folder. Open data.component.html and paste the code into the file. It should look like:

```
<p>
  {{country.name}}
  Population: {{country.population}}
  Location: {{country.latlng}}
</p>
```

Finally, go to the .css file and change the font size and color to observe how the scope of the css file only applies to its component:

```
p{
  color:green;
  font-size: 14px;
}
```

Now save everything and make sure it is loading properly.

CONCLUSION

While this tutorial is not an exhaustive tour of Angular, it is meant to display some of the basic building blocks that make it one of the most popular web frameworks today. Thank you for reading!