Throughout this Angular *6* tutorial, we'll learn to build a full-stack example web application with Angular *6*, the latest version of Angular — The most popular framework/platform for building mobile and desktop client side applications, created and used internally by Google.

Angular 6 Tutorial: Full-Stack Angular Example Application

By the end of this Angular *6* tutorial, you'll learn by building a real world example application:

- how to install the latest version of Angular CLI,
- how to use the Angular *6* CLI to generate a new Angular *6* project,
- how to use Angular *6* to build a simple CRM application,
- what's a component and component-based architecture
- how to create Angular *6* components,
- how to add component routing and navigation,
- how to use **HttpClient** to consume a REST API etc.

The Django Example Back-End

We'll make use of a simple CRM API built with Django and Django REST framework. Since this is an Angular tutorial we'll not focus on building the API as this will be the subject of a separate tutorial but you can grab the source code of the back-end API from this repository

You can use the following commands to start the development server:

```
# Clone the project and navigate into it
$ git clone https://github.com/techiediaries/django-crm
$ cd django-crm

# Create a virtual environment and install packages
$ pipenv install

# Activate the virtual environment
$ pipenv shell

# Create and migrate the database then run the local development server
```

```
$ python manage.py migrate
$ python manage.py runserver
```

You server will be running from `http://localhost:8000`.

We are using pipenv, the officially recommended package management tool for Python so you'll need to have it installed. The process is quite simple depending on your operating system.

Installing the Angular CLI (v6.0.0)

Make sure you have Node.js installed, next run the following command in your terminal to install Angular CLI **v 6.0.0**.

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

You can check the installed version by running the following command:

```
ng version
```

This is the output I'm getting:

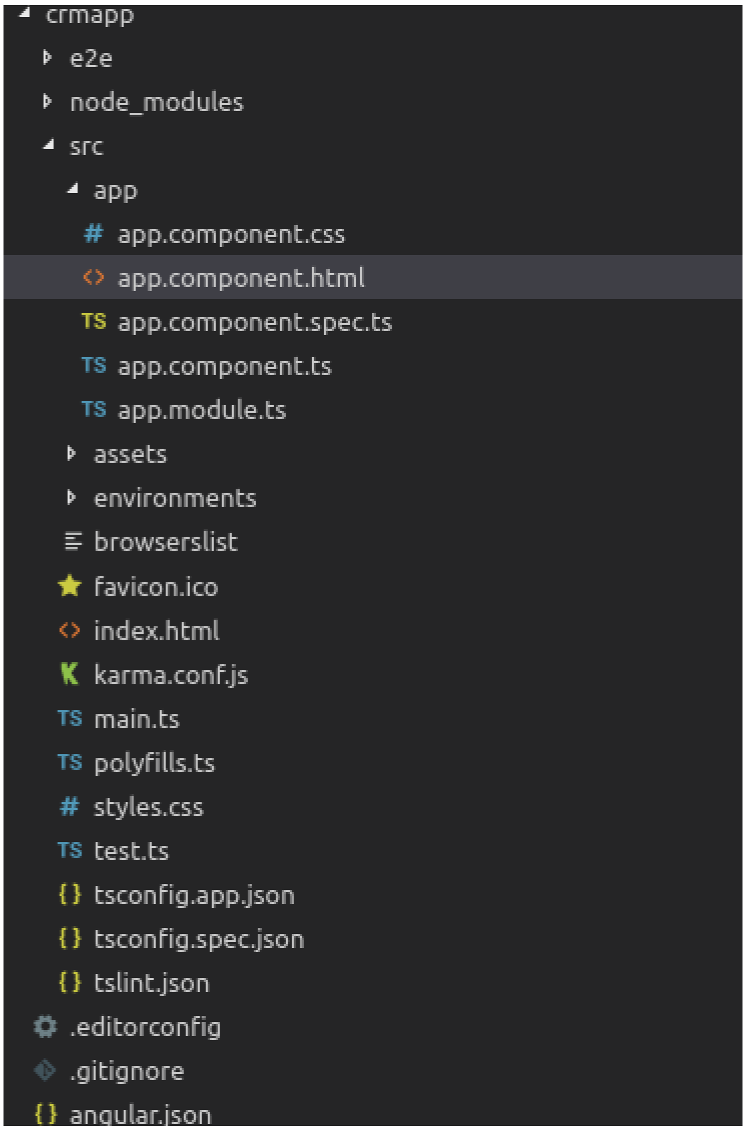
```
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 6.0.0
Node: 8.11.1
OS: linux x64
Angular:
...

Package                           Version
------------------------------------------------------
@angular-devkit/architect         0.6.0
@angular-devkit/core              0.6.0
@angular-devkit/schematics        0.6.0
@schematics/angular               0.6.0
@schematics/update                0.6.0
rxjs                              6.1.0
typescript                        2.7.2
```

Now, you're ready to create a project using Angular CLI *v6*. Simply run the following command in your terminal:

```
ng new crmapp
```

The CLI will automatically generate a bunch of files common to most Angular 6 projects and install the required dependencies for your project.

We will mostly be working inside the `src/app` folder. This is the directory structure of the project:



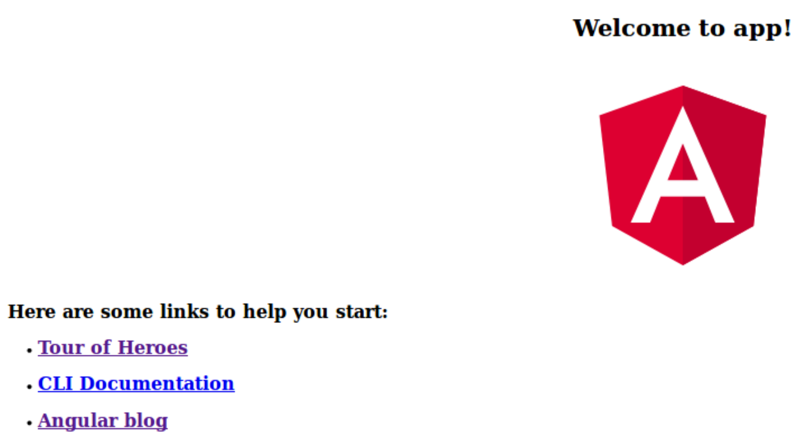
```
crmapp
  e2e
  node_modules
  src
    app
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    assets
    environments
    browserslist
    favicon.ico
    index.html
    karma.conf.js
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    tslint.json
  .editorconfig
  .gitignore
  angular.json
```

You can serve your application locally by running the following commands:

```
# Navigate inside your project's folder
$ cd crmapp

# Serve your application
$ ng serve
```

You application will be running from `http://localhost:4200`.

This is a screen-shot of home page of the application:


Welcome to app!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

Components in Angular 6|5|4

Now what's a component?

A component is a TypeScript class with an HTML template and an optional set of CSS styles that control a part of the screen.

Components are the most important concept in Angular 6. An Angular 6 application is basically a tree of components with a root component (the famous *AppComponent*). The root component is the one contained in the bootstrap array in the main `NgModule` module `app.module.ts`.

One important aspect of components is re-usability. A component can be re-used throughout the application and even in other applications. Common and repeatable code that performs a certain task can be encapsulated into a re-usable component that can be called whenever we need the functionality it provides.

> Each bootstrapped component is the base of its own tree of components. Inserting a bootstrapped component usually triggers a cascade of component creations that fill out that tree. source

Component-Based Architecture

An Angular application is made of several components forming a tree structure with parent and child components.

A component is an independent block of a big system (web application) that communicates with the other building blocks (components) of the system using inputs and outputs. A component has associated view, data and behavior and may have parent and child components.

Components allow maximum re-usability, easy testing, maintenance and separation of concerns.

Let's now see this practically. Head over to your Angular application project folder and open the `src/app` folder. You will find the following files:

- `app.component.css`: the CSS file for the component
- `app.component.html`: the HTML view for the component
- `app.component.spec.ts`: the unit tests or spec file for the component
- `app.component.ts`: the component code (data and behavior)
- `app.module.ts`: the application main module

Except for the last file which contains the declaration of the application main (root) **Module**, all these files are used to create a component. It's the **AppComponent**: The root component of our application. All other components we are going to create next will be direct or un-direct children of the root component.

Demystifying the AppComponent (The Root Component of Angular Applications)

Go ahead and open the `src/app/app.component.ts` file and let's understand the code behind the main/root component of the application.

First, this is the code:

```
import { Component } from  '@angular/core';
@Component({
        selector:  'app-root',
        templateUrl:  './app.component.html',
        styleUrls: ['./app.component.css']
})
export  class  AppComponent {
        title  =  'app';
}
```

We first import the Component decorator from `@angular/core` then we use it to decorate the TypeScript class *AppComponent*. The Component decorator takes an object with many parameters such as:

- *selector*: specifies the tag that can be used to call this component in HTML templates just like the standard HTML tags
- *templateUrl*: indicates the path of the HTML template that will be used to display this component (you can also use the *template* parameter to include the template inline as a string)
- *styleUrls*: specifies an array of URLs for CSS style-sheets for the component

The *export* keyword is used to export the component so that it can be imported from other components and modules in the application.

The *title* variable is a member variable that holds the string 'app'. There is nothing special about this variable and it's not a part of the canonical definition of an Angular component.

Now let's see the corresponding template for this component. If you open `src/app/app.component.html` this is what you'll find:

```
<div  style="text-align:center">
<h1>
Welcome to {{ title }}!
</h1>
        <img  width="300"  alt="Angular Logo"
src="data:image/svg+xml;....">
</div>

        <h2>Here are some links to help you start: </h2>
<ul>
```

```
        <li>
        <h2><a  target="_blank"  rel="noopener"
href="https://angular.io/tutorial">Tour of Heroes</a></h2>
        </li>
        <li>
        <h2><a  target="_blank"  rel="noopener"
href="https://github.com/angular/angular-cli/wiki">CLI Documentation</a>
</h2>
        </li>
        <li>
        <h2><a  target="_blank"  rel="noopener"
href="https://blog.angular.io/">Angular blog</a></h2>
        </li>
</ul>
```

The template is a normal HTML file (almost all HTML tags are valid to be used inside Angular templates except for some tags such as `<script>`, `<html>` and `<body>` etc.) with the exception that it can contain template variables (in this case the *title* variable) or expressions (`{{...}}`) that can be used to insert values in the DOM dynamically. This is called **interpolation** or **data binding**. You can find more information about templates from the docs.

You can also use other components directly inside Angular templates (via the selector property) just like normal HTML.

If you are familiar with the MVC (Model View Controller) pattern, the component class plays the role of the Controller and the HTML template plays the role of the View.

Angular 6 Components by Example

After getting the theory behind Angular components, let's now create the components for our simple CRM application.

Our REST API, built with Django, exposes these endpoints:

- `/api/accounts`: create or read a paginated list of accounts
- `/api/accounts/<id>`: read, update or delete an account
- `/api/contacts`: create or read a paginated list of contacts
- `/api/contacts/<id>`: read, update or delete a contact
- `/api/leads`: create or read a paginated list of leads
- `/api/leads/<id>`: read, update or delete a lead
- `/api/opportunities`: create or read a paginated list of opportunities
- `/api/opportunities/<id>`: read, update or delete an opportunity

Before adding routing to our application we first need to create the application's components so based on the exposed REST API architecture we can initially divide our application into these components:

- `AccountListComponent`: this component displays and controls a tabular list of accounts
- `AccountCreateComponent`: this component displays and controls a form for creating or updating accounts
- `ContactListComponent`: displays a table of contacts
- `ContactCreateComponent`: displays a form to create or update a contact
- `LeadListComponent`: displays a table of leads
- `LeadCreateComponent`: displays a form to create or update a lead
- `OpportunityListComponent`: displays a table of opportunities
- `OpportunityCreateComponent`: displays a form to create or update an opportunity

Let's use the Angular CLI to create the components

```
ng generate component AccountList
ng generate component AccountCreate

ng generate component ContactList
ng generate component ContactCreate

ng generate component LeadList
ng generate component LeadCreate

ng generate component OpportunityList
ng generate component OpportunityCreate
```

This is the output of the first command:

```
CREATE src/app/account-list/account-list.component.css (0 bytes)
CREATE src/app/account-list/account-list.component.html (31 bytes)
CREATE src/app/account-list/account-list.component.spec.ts (664 bytes)
CREATE src/app/account-list/account-list.component.ts (292 bytes)
UPDATE src/app/app.module.ts (418 bytes)
```

You can see that the command generates all the files to define a component and also updates `src/app/app.module.ts`.

If you open `src/app/app.module.ts` after running all commands, you can see that all components are automatically added to the *AppModule* `declarations` array.:

```
import { BrowserModule } from  '@angular/platform-browser';

import { NgModule } from  '@angular/core';
import { AppComponent } from './app.component';

import { AccountListComponent } from './account-list/account-
list.component';

import { AccountCreateComponent } from './account-create/account-
create.component';

import { ContactListComponent } from './contact-list/contact-
list.component';

import { ContactCreateComponent } from './contact-create/contact-
create.component';

import { LeadListComponent } from './lead-list/lead-list.component';

import { LeadCreateComponent } from './lead-create/lead-create.component';

import { OpportunityListComponent } from './opportunity-list/opportunity-
list.component';

import { OpportunityCreateComponent } from './opportunity-
create/opportunity-create.component';

@NgModule({

declarations: [
        AppComponent,
        AccountListComponent,
        AccountCreateComponent,
        ContactListComponent,
        ContactCreateComponent,
        LeadListComponent,
        LeadCreateComponent,
        OpportunityListComponent,
        OpportunityCreateComponent
],
imports: [
        BrowserModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

If you are creating components manually, you need to make sure to include manually so they can be recognized as part of the module.

Adding Angular 6 Routing

Angular CLI provides the `--routing` switch (`ng new crmapp --routing`) that enables you to add routing automatically but we're going to add routing manually for the sake of understanding the various pieces involved in adding component routing to your Angular application.

In fact, adding routing is quite simple:

- add a separate module (which can be called `AppRoutingModule`) in a file `app-routing.module.ts`, and import the module by including it in the `imports` of main `AppModule`,
- add `<router-outlet></router-outlet>` in `app.component.html` (this is where the Angular Router will insert components matching the current path),
- add routes (each route is an object with properties such as *path* and *component* etc.).

This is the initial content of `app-routing.module.ts`:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

The *routes* will contain all the routes of the application. After creating the components we'll see how to add routes to this array.

For now, we want to redirect the visitor to the `/accounts` path when the home URL is visited so the first path we'll add is:

```
{ path:  '', redirectTo:  'accounts', pathMatch:  'full' },
```

The *pathMatch* specifies the matching strategy. `full` means that we want to fully match the path.

Next let's add the other paths:

```
{ path:  '', redirectTo:  'accounts', pathMatch:  'full' },
{
        path:  'accounts',
        component:  AccountListComponent
},
{
        path:  'create-account',
        component:  AccountCreateComponent
},
{
        path:  'contacts',
        component:  ContactListComponent
},
{
        path:  'create-contact',
        component:  ContactCreateComponent
},
{
        path:  'leads',
        component:  LeadListComponent
},
{
        path:  'create-lead',
        component:  LeadCreateComponent
},
{
        path:  'opportunities',
        component:  OpportunityListComponent
},
{
        path:  'create-opportunity',
        component:  OpportunityCreateComponent
}

];
```

Now open `src/app/app.module.ts` and import the routing module then add it to the *imports* array:

```
import {AppRoutingModule} from  './app-routing.module';
[...]

@NgModule({

declarations: [

AppComponent,
[...]
],

imports: [
        BrowserModule,
        AppRoutingModule
],
[...]
})
export  class  AppModule { }
```

Finally, open `src/app/app.component.html` then add the navigation links and the router outlet:

```
<a [routerLink]="'/accounts'"> Accounts </a>
<a [routerLink]="'/create-account'"> Create Account </a>

<a [routerLink]="'/contacts'"> Contacts </a>
<a [routerLink]="'/create-contact'"> Create Contact </a>

<a [routerLink]="'/leads'"> Leads </a>
<a [routerLink]="'/create-lead'"> Create Lead </a>

<a [routerLink]="'/opportunities'"> Opportunities </a>
<a [routerLink]="'/create-opportunity'"> Create Opportunity </a>

<div>
        <router-outlet></router-outlet>
</div>
```

An Example for Consuming the REST API Using Angular 6 HttpClient

Now that we've created the different components and added routing and navigation, let's see an example of how to use the *HttpClient* of Angular 6 to consume the RESTful API back-end.

First, you need to add the *HttpClientModule* module to the *imports* array of the main application module

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

@NgModule({

declarations: [
..
],

imports: [

[..]

HttpClientModule
],
providers: [],
bootstrap: [AppComponent]

})

export  class  AppModule { }
```

Create an Angular 6 Service/Provider

A service is a global class that can be injected in any component. It's used to encapsulate code that can be common between multiple components in one place instead of repeating it throughout various components.

Now, lets create a service that encapsulates all the code needed for interacting with the REST API. Using Angulat CLI run the following command:

```
ng g service api
```

Two files: `src/app/api.service.ts` and `src/app/api.service.spec.ts` will be generated. The first contains code for the service and the second contains tests.

Open `src/app/api.service.ts` then import and inject the *HttpClient* class.

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

@Injectable({
providedIn:  'root'
})

export  class  APIService {

        constructor(private  httpClient:  HttpClient) {}

}
```

Angular 6 provides a way to register services/providers directly in the `@Injectable()` decorator by using the new `providedIn` attribute. This attribute accepts any module of your application or `'root'` for the main app module. Now you don't have to include your service in the *providers* array of your module.

Getting Contacts/Sending HTTP GET Request Example

Let's start with the contacts API endpoint.

- First we'll add a method to consume this endpoint in our global API service,
- next we'll inject the API service and call the method from the corresponding component class (`ContactListComponent`)
- and finally we'll display the result (the list of contacts) in the component template.

Open `src/app/api.service.ts` and add the following method:

```
export  class  APIService {
API_URL  =  'http://localhost:8000';
constructor(private  httpClient:  HttpClient) {}
getContacts(){
        return  this.httpClient.get(`${this.API_URL}/contacts`);
}
```

Next, open `src/app/contact-list/contact-list.component.ts` and inject the *APIService* then call the *getContacts()* method:

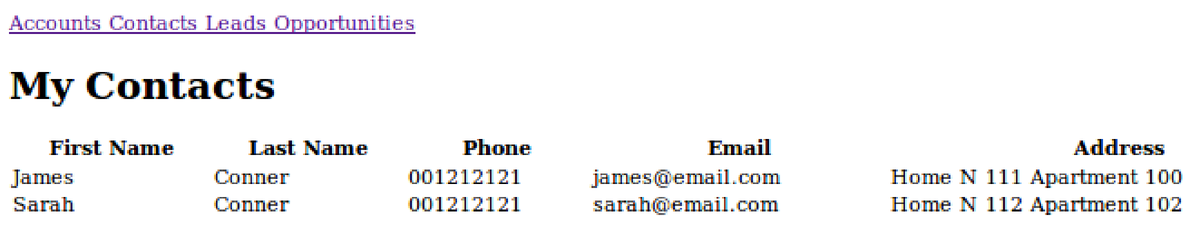
```
import { Component, OnInit } from  '@angular/core';
import { APIService } from  '../api.service';

@Component({
        selector:  'app-contact-list',
        templateUrl:  './contact-list.component.html',
        styleUrls: ['./contact-list.component.css']
})

export  class  ContactListComponent  implements  OnInit {

private  contacts:  Array<object> = [];
constructor(private  apiService:  APIService) { }
ngOnInit() {
        this.getContacts();
}
public  getContacts(){
        this.apiService.getContacts().subscribe((data:  Array<object>) => {
                this.contacts  =  data;
                console.log(data);
        });
}
}
```

Now let's display the contacts in the template. Open `src/app/contact-list/contact-list.component.html` and add the following code:

```
<h1>
My Contacts
</h1>
<div>
<table  style="width:100%">
<tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Phone</th>
        <th>Email</th>
        <th>Address</th>
</tr>
<tr *ngFor="let contact of contacts">
        <td> { { contact.first_name } } </td>
        <td> { { contact.last_name } } </td>
        <td> { { contact.phone } } </td>
        <td> { { contact.email } } </td>
        <td> { { contact.address } } </td>
</tr>
</table>
```

```
</div>
```

This is a screen-shot of the component:



Accounts Contacts Leads Opportunities

My Contacts

First Name	Last Name	Phone	Email	Address
James	Conner	001212121	james@email.com	Home N 111 Apartment 100
Sarah	Conner	001212121	sarah@email.com	Home N 112 Apartment 102

Creating Contacts/Sending HTTP POST Request Example

Now let's create a method to send HTTP Post request to create a random contact. Open the API service file and add the following method:

```
createContact(contact){
        return  this.httpClient.post(`${this.API_URL}/contacts/`,contact);
}
```

Next let's call this method from the `ContactCreateComponent` to create a contact. First open `src/app/contact-create/contact-create.component.ts` and add the following code:

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


@Component({

selector:  'app-contact-create',

templateUrl:  './contact-create.component.html',

styleUrls: ['./contact-create.component.css']

})

export  class  ContactCreateComponent  implements  OnInit {
constructor(private  apiService:  APIService) { }

ngOnInit() {}

createContact(){

var  contact  = {
        account:  1,
        address:  "Home N 333 Apartment 300",
        createdBy:  1,
        description:  "This is the third contact",
        email:  "abbess@email.com",
        first_name:  "kaya",
        isActive: true,
        last_name: "Abbes",
        phone: "00121212101"
};
this.apiService.createContact(contact).subscribe((response) => {
        console.log(response);
});
};
}
}
```

For now, we're simply hard-coding the contact info for the sake of simplicity.

Next open `src/app/contact-create/contact-create.component.html` and add a button to call the method to create a contact:

```
<h1>
Create Contact
</h1>
<button (click)="createContact()">
        Create Contact
</button>
```

Conclusion

Throught this Angular 6 tutorial, we've seen , by building a simple real world example, how to use different Angular concepts to create simple full-stack application with Angular and Django. You can find the source code in this repository.