What is Angular?

Sainathan Iyer(isainath@ford.com)

Angular is a platform for building single page client applications using HTML and TypeScript.

Great for single page applications.

Angular has built-in features like Dependency Injection and Data Binding.

We will use **SpringBoot** to develop REST Web Services and **Microservices**.

Key Takeaways from this Project:

1) Learn how to develop a CRUD Full Stack App with Angular Front End and

Sping Boot Back End.

2) Learn how to build REST APIs with Spring Boot.

3) Learn to solve the challenges of connecting an Angular Front-End to

a RESTful API Back-End.

4) We will learn to connect a REST API to DB using JPA/Hibernate using Spring Boot.

5) Learn the basics of Angular:

- Modules
- Components
- Data Binding
- Routing

Here we follow a Client Server Architecture:

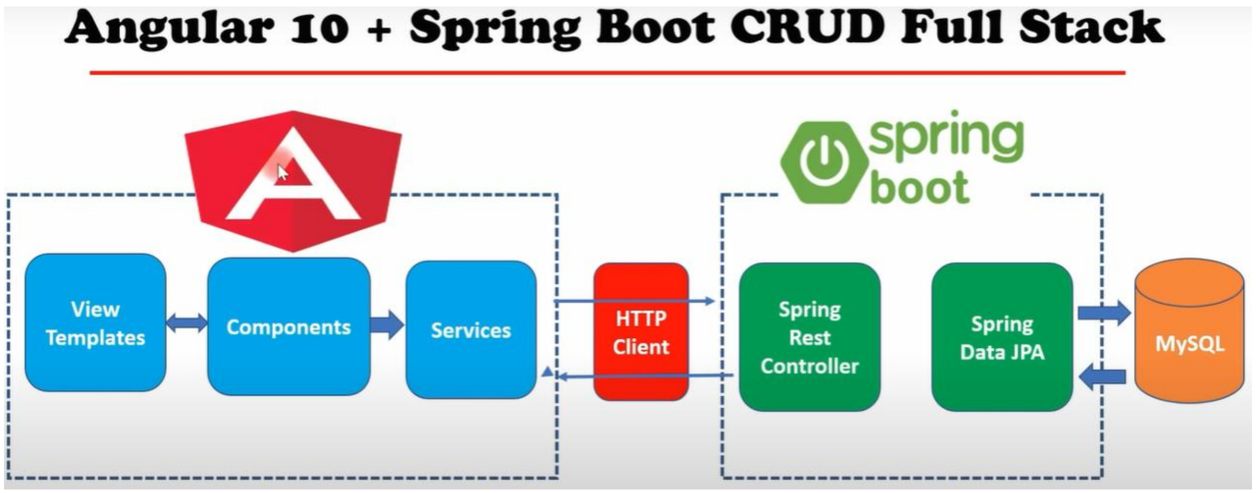
```
Angular App                 Spring Boot App

           --------------------->

 (Client)  <--------------------- (Server)
```

At **server** side we have the **Spring Boot Application** which **creates and Exposes REST APIs.**

At **client** side we have an **Angular Application**, which **consumes the REST APIs** exposed by the Spring Boot Application.



Backend:

We are going to use **DB2** Database. Spring Data JPA will communicate with DB2 Database.

Spring Data JPA internally uses Hibernate as a JPA Provider. And it will provide all the CRUD operations for a particular Entity. Here we will create an Employee Entity. So Spring Data JPA helps to reduce a lot of boiler plate code and helps in developing a Persistence Layer.

Spring REST Controller we develop using **Spring MVC**, to develop the REST Endpoints. Using this we will develop 5 REST APIs for our Employee Resource.

Frontend:

Let's take a look at the Angular Architecture, here we have different components. So here we have View Templates, Components and Services. We may also have Directives and Pipes also.

View Templates: We use **HTML** to develop Templates.

Components: It contains **properties** and **methods**, which will handle data for templates. We can perform **2 Way Binding** between **Templates and Components**. Components will basically handle the User Interface Data.

Services: They are Angular Services where we keep all the common logic, and we **inject** Services in the required Components using Angular **Dependency Injection**. We write all the **REST Client** code in Services to make a **REST API call**.

Angular has its own **HTTP Client Module** to make a REST API call.

Tools and Technologies used in Front-End:

Angular: Web Development Framework

TypeScript: To write code in the Angular Application

NodeJS and **NPM**: JavaScript runtime environment for running the Front-End

VS Code: Integrated Development Environment

Angular CLI: To generate components, services and classes

Bootstrap: CSS Library for styling the Angular App

Project Development Process:

List, Create, Update, Delete and View Functionalities are implemented for Employees.

To create a new project using Angular CLI we use the command:

ng new angular-frontend

Angular Folder Structure:

1) **package.json**: It contains name of the project, version of the project, few scripts, dependencies(tools, libraries and packages) required to run the Angular Application.
2) When we perform **"npm install"** it will install all the dependencies in the **node_modules** folder.
3) Then we have **tsconfig.app.json and tsconfig.spec.json** files which are used to convert the TS code to JS code, because Browser doesn't understand TS code. Browsers can't execute TypeScript code directly, so it should be **"Transpiled"** into JavaScript using the **tsc** compiler.
4) **node_modules** Folder: This folder contains all the dependencies and packages required to run the Angular Application.
5) **src** Folder: All the development code is inside this folder.
   a) **main.ts**: It's the entry point of the Angular Application.
   b) **index.html**: This is the single file which will get served in the browser. Since we are developing a single page application we should have a single html file. In this file we have **<app-root>**. This is the selector which we have configured in the **app-component.ts** file. And this selector itself calls the **app-component.html** template file which we see as the cover page of the app when we run **ng serve** command.

```
c) @Component({
d)   selector: 'app-root',
e)   templateUrl: './app.component.html',
f)   styleUrls: ['./app.component.css']
g) })
```

c) **app.module.ts**: This is the root module of our application. In this module we configure the **components** in the **declarations** array and **dependent** modules inside the **imports** array. We also configure the **providers** like **services**. And within the app.module.ts file we configure the **bootstrap** and kick start our **AppComponent** which is our root component.

d) Angular application can have any number of modules, but it should have at least one module and that is called as the **AppModule**.

e) Angular application can have any number of components, but it should have at least one component and that is called as the **AppComponent**.

f) **polyfills.ts**: Polyfills in angular are few lines of code which make your application compatible for different browsers.

g) **styles.css**: Here we basically globally configure the css files.

h) **assets** Folder: It contains the static files like images.

i) **app-routing.module.ts**: Here we basically configure the routing of our angular application.

How the Control Flows in our Angular Application:

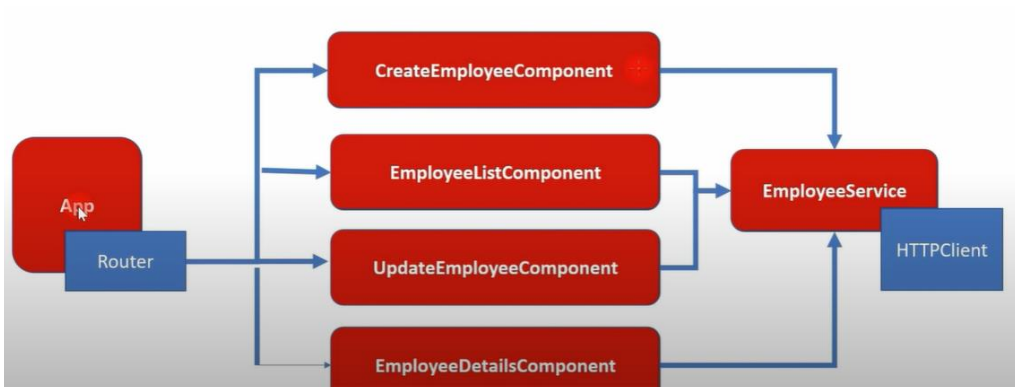
So **main.ts** file is the entry point of our application and this file bootstraps and kick starts the **AppModule** using **bootstrapModule()** method.

AppModule bootstraps and kick starts the **AppComponent**.

AppComponent has a property called as "**title**", and this title will be rendered inside the **app.compoent.html** template.

This is how the control flows in Angular Application.

Angular App Components Overview:



1) Here we are going to create 4 components with respect to their functionalities.
2) We have the **EmployeeService** which is an Angular Service which contains all the **HTTP REST client code**.
3) We can **Inject** the **EmployeeService** in the **required components** using **Angular Dependency Injection** which is used to inject angular services in various components.
4) For example we can keep all the common logging-in logic in the EmployeeService and inject it in the required components and call the logging-in methods.
5) So in our app we create an **EmployeeService class** which will **communicate with the server** using the **REST APIs**.
6) **EmployeeService** internally uses **HttpClient** module to **make a REST API call.**
7) So basically we are going to make a **GET, PUT, POST** and **DELETE** REST API calls. So we write the **REST Client Code** in **EmployeeService** to **make the REST API calls**.
8) We are going to configure **Routing** in **AppRouting** Module. Angular provides its own Routing Module.
9) We are also going to create an **Employee Model**, it's basically a TypeScript class, it contains **properties** and we create this **Employee class model** to **hold the response of the REST APIs**.
10) **{{**title**}}** is called as **Interpolation**, to use **properties** of a **Component**.
11) Now we create a **EmployeeListComponent**, which will display a List of Employees on a Webpage. For that we first create an Employee class using the command : **ng g class employee**
12) Now we define some **properties** of Employee in the **employee.ts** class file which will hold the response of the REST APIs.
13) Now we will create **employee-list** Component. This command also updates the **app.module.ts** file by adding **EmployeeListComponent** in the **declarations** array.
14) So this component now belongs to **AppModule**.
15) **employee-list.component.ts** will basically handle the data and process it. And it's annotated with **@Component** decorator. And this decorator has 3 attributes (selector, templateUrl, styleUrls).
16) To run the **employee-list.component.html** we copy the selector **app-employee-list** from **employee-list.component.ts** file and paste it in the **app.component.html**.
17) Inside a table row **<tr>** we are going to use ***ngFor** inbuilt Angular Directive to iterate over List of Employees. (iterate over HTML elements)

18) **Angular Service**: We use angular service to share data among various components in Angular App. **Service** is responsible to **fetch the data from server**.

19) To **define a class as a Service** in Angular we use **@Injectable** decorator to provide the metadata that allows angular to inject into a component as a dependency.

20) To generate a Service we give the following command:
ng g s employee

21) This will create 2 files:
   a) employee.service.ts
   b) employee.service.spec.ts

22) The @Injectable decorator marks the employee service class as a provider which can be injected into various components.

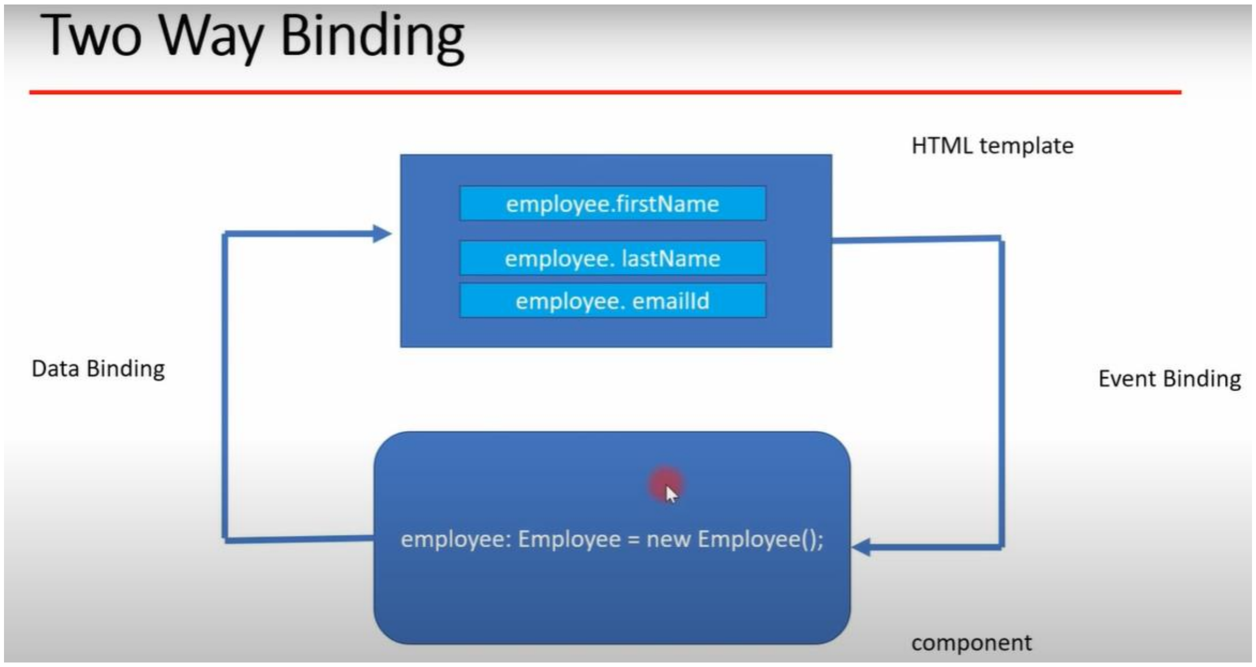
23) Once we inject Employee Service in Employee List Component then we can call the service methods in the component file.

24) Add the `@CrossOrigin(origins = "http://localhost:4200")` line at the top of the Controller File in Spring Boot Backend to avoid the **CORS** error.

25) We need to configure the Routes in the **app-routing.module.ts**.

26) **Routes** has 2 properties **path** and **component** to be configured.
   a) **path**: Specifies the **URL** path for the Route
   b) **component**: Specifies the **Component** to be displayed for the given path.

27) Two Way Binding in Angular:



Consider we have a form with firstName, lastName and email. And similarly we have a component class “employee” which has the properties firstName, lastName and email. So we access these properties in the HTML template using employee.firstName, employee.lastName and employee.email.

Whenever we **enter something in** employee.firstName, employee.lastName and employee.email **input controls** then the **corresponding properties** will get **automatically updated in component class**. And whenever we do some changes in the **component class**, the **corresponding properties will get updated** in the **HTML template**, so this is called as **the Two Way Data Binding in Angular**.

We can use **ngModel** directive to achieve **Two Way Binding** in our Angular Application.

For creating an Employee we need to create a **create-employee** component using the following command: **ng g c create-employee**

Then we have to create a Create Employee Form in **create-employee.component.html** to Add a new Employee to the Employee List.

Inside the <form> here, we are using the **ngSubmit** to **listen to the Form Submission Event**. And we assign an Event Handler **onSubmit()** to the ngSubmit event. And we define this event handler in the component class.

In this we make use of **[(ngModel)]** directive for two way binding.

Now we need to make a REST API call and send the form data to backend to store in the database.

And after **successfully saving the data**, it should get **redirected** to the **Employees List Component**. So here we will us the **Router to Navigate** to the Employee List Page.

We use **ActivatedRoute** module to retrieve **id** from the route.