

Practical

Angular 8 with SpringBoot

By

Stanley Stephen

Linkedin: <https://www.linkedin.com/in/contactstanley/>

Github: <https://github.com/stanleymca>

Email: s.stanley.mca@gmail.com

Angular 8 with SpringBoot - CRUD Implementation

We are going to cover the architecture and the following topics in this Angular 8 with SpringBoot – CRUD sample implementation.

1. Topic 1 - Develop SpringBoot CRUD Rest APIs

- Creating and Importing a Project
- Packaging Structure
- The pom.xml File
- Configuring Database (H2 / MySQL)
- Create JPA Entity - Employee.java
- Create a Spring Data Repository - EmployeeRepository.java
- Create Spring Rest Controller - EmployeeController.java
- Exception (Error) Handling for RESTful Services
- Running the Application
- Testing REST APIs
 - Create an employee
 - List all employees
 - Get employee by id
 - Update an employee
 - Delete an employee

2. Topic 2 - Create Angular 8 App

- Install the latest version of Angular CLI
- Create Angular 8 client application using Angular CLI
- Identify Components, Services, and Modules
- Create Service & Components using Angular CLI
- Integrate JQuery and Bootstrap with Angular

3. Topic 3 - Develop Angular 8 CRUD Operations

- Create an Employee class
- Employee Service
- Creating Employee List Template and Component
- Create Add Employee Template and Component
- Update Employee Template and Component
- Create View Employee Details Template and Component

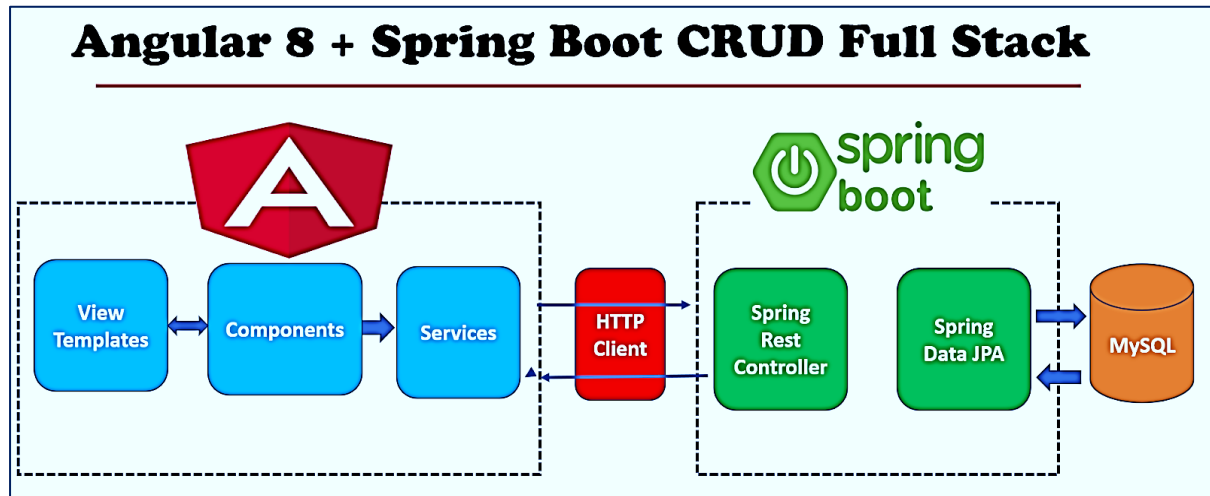
4. Topic 4 - Angular 8 CRUD App Configuration

- npm package.json - Configure Dependencies
- App Routing Module
- App Component
- App Component Template
- App Module
- Main Index Html File
- Main (Bootstrap) File
- Polyfills
- TypeScript tsconfig.json

5. Topic 5 - Running Angular 8 CRUD App

- Running Angular 8 Client Application
- Demo

Angular 8 - Spring Boot CRUD App Architecture



Implementation of Features

- Create an Employee
- Update an Employee
- List of Employees
- Delete an Employee
- View an Employee

What we will build?

Basically, we will create two projects:

1. **springboot2-jpa-crud-example:** This project is used to develop CRUD RESTful APIs for a simple **Employee Management System** using SpringBoot 2, JPA and MySQL as a database.
2. **angular8-springboot-client:** This project is used to develop single page application using Angular 8 as front-end technology. This Angular 8 application consumes CRUD Restful APIs developed and exposed by a **springboot2-jpa-crud-example** project.

Topic 1 - Develop SpringBoot CRUD Rest APIs

Spring Boot CRUD Rest APIs Development

Following are five REST APIs (Controller handler methods), we will develop for *Employee* resource.

Sr. No.	API Name	HTTP Method	Path	Status Code	Description
(1)	GET Employees	GET	/api/v1/employees	200 (OK)	All Employee resources are fetched.
(2)	POST Employee	POST	/api/v1/employees	201 (Created)	A new Employee resource is created.
(3)	GET Employee	GET	/api/v1/employees/{id}	200 (OK)	One Employee resource is fetched.
(4)	PUT Employee	PUT	/api/v1/employees/{id}	200 (OK)	Employee resource is updated.
(5)	DELETE Employee	DELETE	/api/v1/employees/{id}	204 (No Content)	Employee resource is deleted.

1. Creating and Importing a Project

There are many ways to create a Spring Boot application. The simplest way is to use Spring.

The screenshot shows the Spring Initializr interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there are dropdown menus to "Generate a" (Maven Project), "with" (Java), and "and Spring Boot" (2.0.5). The "Project Metadata" section has "Artifact coordinates" with "Group" set to "net.guides.springboot2" and "Artifact" set to "springboot2-jpa-crud-example". The "Dependencies" section has a search bar with "Web, Security, JPA, Actuator, Devtools..." and "Selected Dependencies" showing "Web", "DevTools", "MySQL", and "JPA". A "Generate Project" button is at the bottom.

Initializr at <http://start.spring.io/>, which is an online Spring Boot application generator.

We have specified the following details in the Spring Initializr as shown in the above image:

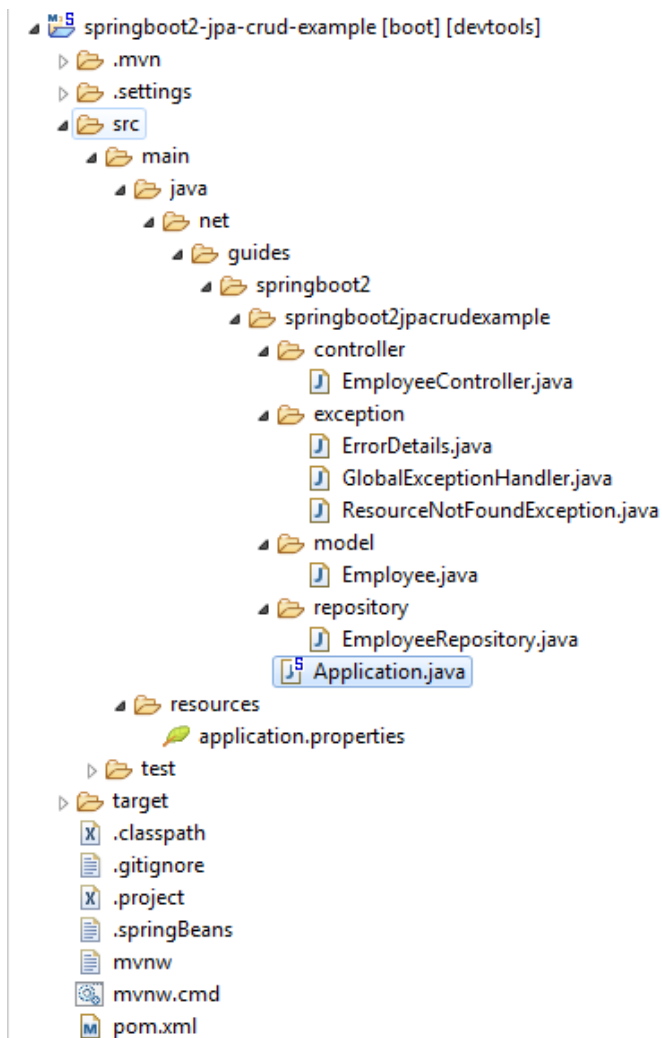
- **Generate:** Maven Project
- **Java Version:** 1.8 (Default)
- **SpringBoot:** 2.0.4
- **Group:** net.guides.springboot2

- **Artifact:** springboot2-jpa-crud-example
- **Name:** springboot2-jpa-crud-example
- **Description:** Rest API for a Simple Employee Management Application
- **Package Name:** net.guides.springboot2.springboot2jpacrudexample
- **Packaging:** jar (This is the default value)
- **Dependencies:** Web, JPA, MySQL, DevTools

Once, all the details are entered, click on Generate Project button will generate a spring boot project and downloads it. Next, Unzip the downloaded zip file and import it into your favourite IDE.

2. Packaging Structure

Following is the packing structure of our **Employee Management System**.



3. The pom.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>net.guides.springboot2</groupId>
  <artifactId>springboot2-jpa-crud-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>springboot2-jpa-crud-example</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

4. Configuring Database (H2 / MySQL)

H2 Database Configuration

In this project, we will use the H2 database to quickly set up and run a spring boot project without installing databases.

Note that we have added below dependency in *pom.xml* file:

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

If you use the H2 database then we no need to configure the database related properties in *application.properties* file.

If you use H2 in-memory database then you no need to create a database and tables, spring boot automatically do it for you.

In this project, we are going to use context path for spring boot project so let's add below property in *application.properties* file:

```
server.servlet.context-path=/springboot-crud-rest
```

MySQL Database Configuration

You can also use the MySQL database but make sure that you follow the below steps to configure MySQL database in this spring boot project.

Step 1: Replace the H2 database dependency with MySQL dependency:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Step 2: Configure *application.properties* to connect to your MySQL database. Add the following content to *application.properties* file:

```
spring.datasource.url = jdbc:mysql://localhost:3306/users_database?useSSL=false
spring.datasource.username = root
spring.datasource.password = root

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

server.servlet.context-path=/springboot-crud-rest
```

Make sure that you will change the above database configuration such as JDBC URL, username and password as per your environment.

Step 3: You need to create a database in MySQL server with the following command:

```
create database users_database
```

Hibernate will automatically create database tables so you only need to manually create the database and configure an *application.properties* file.

5. Create JPA Entity - Employee.java

```
package net.guides.springboot2.springboot2jpacrudexample.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
```

```

@Entity
@Table(name = "employees")
public class Employee {

    private long id;
    private String firstName;
    private String lastName;
    private String emailId;

    public Employee() {
    }

    public Employee(String firstName, String lastName, String emailId) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.emailId = emailId;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    @Column(name = "first_name", nullable = false)
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    @Column(name = "last_name", nullable = false)
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Column(name = "email_address", nullable = false)
    public String getEmailId() {
        return emailId;
    }

    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", firstName=" + firstName + ",
lastName=" + lastName + ", emailId=" + emailId
        + "]";
    }
}

```

6. Create a Spring Data Repository - EmployeeRepository.java

```
package net.guides.springboot2.springboot2jpacrudexample.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import net.guides.springboot2.springboot2jpacrudexample.model.Employee;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}
```

7. Create Spring Rest Controller - EmployeeController.java

```
package net.guides.springboot2.springboot2jpacrudexample.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import net.guides.springboot2.springboot2jpacrudexample.exception.ResourceNotFoundException;
import net.guides.springboot2.springboot2jpacrudexample.model.Employee;
import net.guides.springboot2.springboot2jpacrudexample.repository.EmployeeRepository;

@RestController @CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/v1")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping("/employees")
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }
}
```

```
@GetMapping("/employees/{id}")
public ResponseEntity<Employee> getEmployeeById(@PathVariable(value =
"id") Long employeeId)
    throws ResourceNotFoundException {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not
found for this id :: " + employeeId));
    return ResponseEntity.ok().body(employee);
}

@PostMapping("/employees")
public Employee createEmployee(@Valid @RequestBody Employee employee) {
    return employeeRepository.save(employee);
}

@PutMapping("/employees/{id}")
public ResponseEntity<Employee> updateEmployee(@PathVariable(value =
"id") Long employeeId,
    @Valid @RequestBody Employee employeeDetails) throws
ResourceNotFoundException {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not
found for this id :: " + employeeId));

    employee.setEmailId(employeeDetails.getEmailId());
    employee.setLastName(employeeDetails.getLastName());
    employee.setFirstName(employeeDetails.getFirstName());
    final Employee updatedEmployee = employeeRepository.save(employee);
    return ResponseEntity.ok(updatedEmployee);
}

@DeleteMapping("/employees/{id}")
public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id")
Long employeeId)
    throws ResourceNotFoundException {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not found
for this id :: " + employeeId));

    employeeRepository.delete(employee);
    Map<String, Boolean> response = new HashMap<>();
    response.put("deleted", Boolean.TRUE);
    return response;
}
}
```

Enable CORS on the Server

To enable CORS on the server, add a @CrossOrigin annotation to the EmployeeController

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
```

```
@RequestMapping("/api/v1")
public class EmployeeController {
    // ....
}
```

8. Exception (Error) Handling for RESTful Services

SpringBoot provides a good default implementation for exception handling for RESTful Services. Let's quickly look at the default Exception Handling features provided by SpringBoot.

Resource Not Present

Here's what happens when you fire a request to not resource found: **<http://localhost:8080/some-dummy-url>**

```
{
  "timestamp": 1512713804164,
  "status": 404,
  "error": "Not Found",
  "message": "No message available",
  "path": "/some-dummy-url"
}
```

That's a cool error response. It contains all the details that are typically needed.

What happens when we throw an Exception?

Let's see what Spring Boot does when an exception is thrown from a **Resource**. We can specify the Response Status for a specific exception along with the definition of the Exception with '**@ResponseStatus**' annotation.

Let's create a **ResourceNotFoundException.java** class.

```
package com.companyname.springbootcrudrest.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends Exception{

    private static final long serialVersionUID = 1L;

    public ResourceNotFoundException(String message){
        super(message);
    }
}
```

Customizing Error Response Structure

Default error response provided by Spring Boot contains all the details that are typically needed.

However, you might want to create a framework independent response structure for your organization. In that case, you can define a specific error response structure.

Let's define a simple error response bean.

```
package com.companyname.springbootcrudrest.exception;

import java.util.Date;

public class ErrorDetails {
    private Date timestamp;
    private String message;
    private String details;

    public ErrorDetails(Date timestamp, String message, String details) {
        super();
        this.timestamp = timestamp;
        this.message = message;
        this.details = details;
    }

    public Date getTimestamp() {
        return timestamp;
    }

    public String getMessage() {
        return message;
    }

    public String getDetails() {
        return details;
    }
}
```

To use `ErrorDetails` to return the error response, let's create a `GlobalExceptionHandler` class annotated with `@ControllerAdvice` annotation.

This class handles exception specific and global exception in a single place.

```
package com.companyname.springbootcrudrest.exception;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<?> resourceNotFoundException(ResourceNotFoundException
ex, WebRequest request) {
        ErrorDetails errorDetails = new ErrorDetails(new Date(), ex.getMessage(),
request.getDescription(false));
        return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> globleExcpetionHandler(Exception ex, WebRequest
request) {
        ErrorDetails errorDetails = new ErrorDetails(new Date(), ex.getMessage(),
request.getDescription(false));
        return new ResponseEntity<>(errorDetails,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

9. Running the Application

This spring boot application has an entry point Java class called **Application.java** with the `public static void main(String[] args) method`, which you can run to start the application.

```
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

The `main()` method uses Spring Boot's `SpringApplication.run()` method to launch an application.

```
3* import org.springframework.boot.SpringApplication;[]
5
6 @SpringBootApplication
7 public class Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Application.class, args);
11     }
12 }
13 }
```

Or you can start spring boot application via command line using **mvn spring-boot:run** command.

10. Testing REST APIs

Use below Rest endpoints to test CRUD Rest APIs and in Angular application.

Create an Employee:

HTTP Method - POST

<http://localhost:8080/springboot-crud-rest/api/v1/employees>

List All Employees:

HTTP Method: GET

<http://localhost:8080/springboot-crud-rest/api/v1/employees>

Get an Employee By Id:

HTTP Method GET

<http://localhost:8080/springboot-crud-rest/api/v1/employees/{employeeId}>

Update an Employee

HTTP Method - POST

<http://localhost:8080/springboot-crud-rest/api/v1/employees/{employeeId}>

Delete an Employee By Id:

HTTP Method - DELETE

<http://localhost:8080/springboot-crud-rest/api/v1/employees/{employeeId}>

We have completed the development of SpringBoot CRUD Rest APIs.

Topic 2 - Create Angular 8 App

- Install the latest version of Angular CLI
- Create Angular 8 client application using Angular CLI
- Identify Components, Services, and Modules
- Create Service & Components using Angular CLI
- Integrate JQuery and Bootstrap with Angular

Angular 8 Client App Development

Let's develop a step by step CRUD (Create, Read, Update, Delete) web application using Angular 8 which consumes CRUD rest APIs, which we created in **Part 1**.

I assume that you have installed **Node.js**. Now, we need to check the **Node.js** and NPM versions. Open the terminal or Node command line then type these commands.

```
C:\Angular>node -v
v10.15.3

C:\Angular>npm -v
6.9.0
```

Install the latest version of Angular CLI

To install or update Angular 8 CLI, type this command in the Terminal or Node Command-Line.

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

Now, let's check the latest version of Angular CLI:

```
C:\angular>ng --version
```

The logo for Angular CLI, featuring the word "Angular" in a stylized font with a large "A" and "C", followed by "CLI" in a bold, blocky font.

```
Angular CLI: 8.0.1
Node: 10.15.3
OS: win32 x64
Angular:
```

...

Package	Version

@angular-devkit/architect	0.800.1
@angular-devkit/core	8.0.1
@angular-devkit/schematics	8.0.1
@schematics/angular	8.0.1
@schematics/update	0.800.1
rxjs	6.4.0

Create Angular 8 client application using Angular CLI

The **Angular CLI** is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications.

If you are new to Angular CLI then check out official documentation at <https://cli.angular.io>.

Let's use the below command to generate an Angular 8 Client application. We name this project as "**angular8-springboot-client**".

```
ng new angular8-springboot-client
```

Identify Components, Services, and Modules

Let's list out what are **components**, **services**, and **modules** we are going to create in this application. We will use Angular CLI to generate components, services because Angular CLI follows best practices and saves much of time.

Components

- create-employee
- employee-list
- employee-details

Services

- employee.service.ts - Service for Http Client methods

Modules

- FormsModule
- HttpClientModule
- AppRoutingModule

Employee Class (Typescript class)

- employee.ts: class Employee (id, firstName, lastName, emailId)

In this next step, we will generate these components, classes, and services using **Angular CLI**.

Create Service & Components using Angular CLI

Let's auto-generate the service and components using **Angular CLI**. Change your project directory to **angular8-springboot-client\src\app** and run the following commands:

```
- ng g s employee
- ng g c create-employee
- ng g c employee-details
- ng g c employee-list
```

Here is complete command and output for your reference:

```
C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client>cd src/app

C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client\src\app>ng
g s employee
CREATE src/app/employee.service.spec.ts (343 bytes)
CREATE src/app/employee.service.ts (137 bytes)

C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client\src\app>ng
g c create-employee
CREATE src/app/create-employee/create-employee.component.html (34 bytes)
CREATE src/app/create-employee/create-employee.component.spec.ts (685 bytes)
CREATE src/app/create-employee/create-employee.component.ts (304 bytes)
CREATE src/app/create-employee/create-employee.component.css (0 bytes)
UPDATE src/app/app.module.ts (509 bytes)

C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client\src\app>ng
g c employee-list
CREATE src/app/employee-list/employee-list.component.html (32 bytes)
CREATE src/app/employee-list/employee-list.component.spec.ts (671 bytes)
CREATE src/app/employee-list/employee-list.component.ts (296 bytes)
CREATE src/app/employee-list/employee-list.component.css (0 bytes)
```

```
UPDATE src/app/app.module.ts (617 bytes)

C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client\src\app>ng
g c employee-list
ERROR! src/app/employee-list/employee-list.component.html already exists.
ERROR! src/app/employee-list/employee-list.component.spec.ts already exists.
ERROR! src/app/employee-list/employee-list.component.ts already exists.
ERROR! src/app/employee-list/employee-list.component.css already exists.
The Schematic workflow failed. See above.

C:\Angular\angular8-springboot-crud-tutorial\angular8-springboot-client\src\app>ng
g c employee-details
CREATE src/app/employee-details/employee-details.component.html (35 bytes)
CREATE src/app/employee-details/employee-details.component.spec.ts (692 bytes)
CREATE src/app/employee-details/employee-details.component.ts (308 bytes)
CREATE src/app/employee-details/employee-details.component.css (0 bytes)
UPDATE src/app/app.module.ts (737 bytes)
```

Integrate JQuery and Bootstrap with Angular

Use NPM to download **Bootstrap** & **JQuery**. **Bootstrap** and **jQuery** will be installed into the **node_modules** folder.

```
npm install bootstrap jquery --save
```

Configure installed **Bootstrap** & **JQuery** in an *angular.json* file:

```
...
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
...
```

If bootstrap won't work then try to import bootstrap CSS in style.css like:

```
/* You can add global styles to this file, and also import other style files */

@import '~bootstrap/dist/css/bootstrap.min.css';
```

```
.footer {  
  position: absolute;  
  bottom: 0;  
  width:100%;  
  height: 70px;  
  background-color: blue;  
  text-align: center;  
  color: white;  
}
```

Let's discuss each of the above generated components and service files and we will customize it as per our requirement.

In this topic, we have created and setup the Angular 8 application.

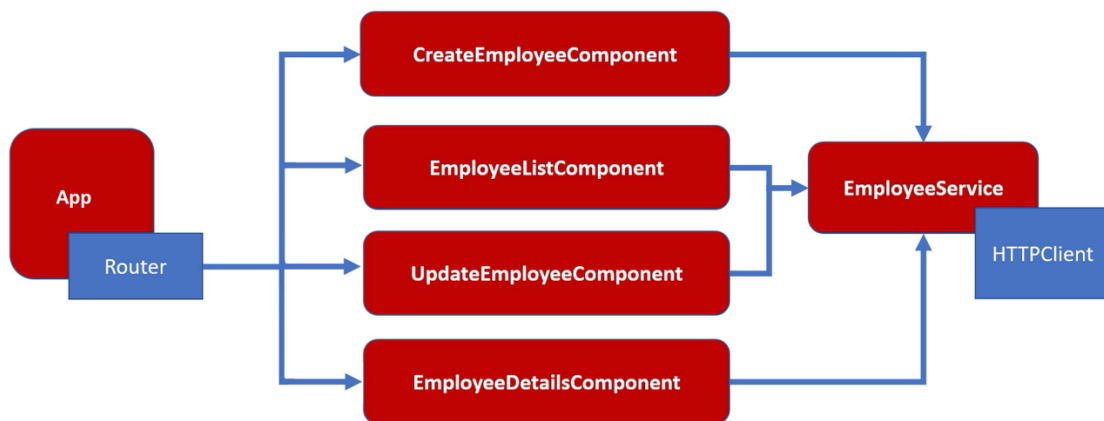
In the next topic, we will create CRUD operations for the Employee model using Angular 8.

Topic 3 - Develop Angular 8 CRUD Operations

1. Create an Employee class
2. Employee Service
3. Creating Employee List Template and Component
4. Create Add Employee Template and Component
5. Update Employee Template and Component
6. Create View Employee Details Template and Component

The below diagram summaries the Angular components that we are going to create going forward:

Angular App Component Diagram



1. Create an Employee Model (TypeScript)

Path - src/app/employee.ts

Before defining the **EmployeeListComponent**, let's define an **Employee** class for working with employees. Create a new file *employee.ts* inside *src/app* folder and add the following code to it.

```
export class Employee {
  id: number;
  firstName: string;
  lastName: string;
  emailId: string;
  active: boolean;
}
```

2. Creating Employee List Template and Component

List All Employee Component

Path - src/app/employee-list/employee-list.component.ts

Let's create **EmployeeListComponent** component which will be used to display a list of employees, create a new employee, and delete an employee.

Update/remove the content of *employee-list.component.ts* inside *src/app* directory and add the following code to it.

```
import { EmployeeDetailsComponent } from '../employee-details/employee-
details.component';
import { Observable } from "rxjs";
import { EmployeeService } from "../employee.service";
import { Employee } from "../employee";
import { Component, OnInit } from "@angular/core";
import { Router } from '@angular/router';

@Component({
  selector: "app-employee-list",
  templateUrl: "../employee-list.component.html",
  styleUrls: ["../employee-list.component.css"]
})
export class EmployeeListComponent implements OnInit {
  employees: Observable<Employee[]>;

  constructor(private employeeService: EmployeeService,
    private router: Router) {}

  ngOnInit() {
    this.reloadData();
  }

  reloadData() {
    this.employees = this.employeeService.getEmployeesList();
  }

  deleteEmployee(id: number) {
    this.employeeService.deleteEmployee(id)
      .subscribe(
        data => {
          console.log(data);
          this.reloadData();
        },
        error => console.log(error));
  }

  employeeDetails(id: number) {
    this.router.navigate(['details', id]);
  }
}
```

List All Employee Template

Path - `src/app/employee-list/employee-list.component.html`

Add *employee-list.component.html* file with the following code to it.

```
<div class="panel panel-primary">
  <div class="panel-heading">
    <h2>Employee List</h2>
  </div>
  <div class="panel-body">
    <table class="table table-striped">
      <thead>
        <tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Email</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let employee of employees | async">
          <td>{{employee.firstName}}</td>
          <td>{{employee.lastName}}</td>
          <td>{{employee.emailId}}</td>
          <td><button (click)="deleteEmployee(employee.id)" class="btn btn-
danger">Delete</button>
          <button (click)="employeeDetails(employee.id)" class="btn
btn-info" style="margin-left: 10px">Details</button>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

3. Create Add Employee Template and Component

Create Employee Component

Path - `src/app/create-employee/create-employee.component.ts`

CreateEmployeeComponent is used to create and handle a new employee form data. Add the following code to it.

```
import { EmployeeService } from '../employee.service';
import { Employee } from '../employee';
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
```



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

  employee: Employee = new Employee();
  submitted = false;

  constructor(private employeeService: EmployeeService,
    private router: Router) { }

  ngOnInit() {
  }

  newEmployee(): void {
    this.submitted = false;
    this.employee = new Employee();
  }

  save() {
    this.employeeService
      .createEmployee(this.employee).subscribe(data => {
        console.log(data)
        this.employee = new Employee();
        this.gotoList();
      },
      error => console.log(error));
  }

  onSubmit() {
    this.submitted = true;
    this.save();
  }

  gotoList() {
    this.router.navigate(['/employees']);
  }
}
```

Create Employee Template

Path - src/app/create-employee/create-employee.component.html

The *create-employee.component.html* shows the add employee HTML form. Add the following code to it.

```
<h3>Create Employee</h3>
<div [hidden]="submitted" style="width: 400px;">
  <form (ngSubmit)="onSubmit()">
```

```
<div class="form-group">
  <label for="name">First Name</label>
  <input type="text" class="form-control" id="firstName" required
  [(ngModel)]="employee.firstName" name="firstName">
</div>
<div class="form-group">
  <label for="name">Last Name</label>
  <input type="text" class="form-control" id="lastName" required
  [(ngModel)]="employee.lastName" name="lastName">
</div>

  <div class="form-group">
    <label for="name">Email Id</label>
    <input type="text" class="form-control" id="emailId" required
    [(ngModel)]="employee.emailId" name="emailId">
  </div>

  <button type="submit" class="btn btn-success">Submit</button>
</form>
</div>

<div [hidden]="!submitted">
  <h4>You submitted successfully!</h4>
  <!-- <button class="btn btn-success" (click)="newEmployee()">Add</button>
-->
</div>
```

4. Update Employee Template and Component

Let's create update employee component with following Angular CLI command:

```
> ng g c update-employee
```

Update Employee Component

Path - *src/app/update-employee/update-employee.component.ts*

UpdateEmployeeComponent is used to update an existing employee.

In this *UpdateEmployeeComponent*, we first get the employee object using REST API and populate in HTML form via data binding. Users can edit the employee form data and submit the form.

Let's add the following code to *UpdateEmployeeComponent*.

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../employee';
import { ActivatedRoute, Router } from '@angular/router';
```

```
import { EmployeeService } from '../employee.service';

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

  id: number;
  employee: Employee;

  constructor(private route: ActivatedRoute, private router: Router,
    private employeeService: EmployeeService) { }

  ngOnInit() {
    this.employee = new Employee();

    this.id = this.route.snapshot.params['id'];

    this.employeeService.getEmployee(this.id)
      .subscribe(data => {
        console.log(data)
        this.employee = data;
      }, error => console.log(error));
  }

  updateEmployee() {
    this.employeeService.updateEmployee(this.id, this.employee)
      .subscribe(data => {
        console.log(data);
        this.employee = new Employee();
        this.gotoList();
      }, error => console.log(error));
  }

  onSubmit() {
    this.updateEmployee();
  }

  gotoList() {
    this.router.navigate(['/employees']);
  }
}
```

Update Employee Template

Path - src/app/update-employee/update-employee.component.html

The update-employee.component.html shows the update employee HTML form. Add the following code to this file.

```
<h3>Update Employee</h3>
```

```
<div style="width: 400px;">
  <form (ngSubmit)="onSubmit()">
    <div class="form-group">
      <label for="name">First Name</label>
      <input type="text" class="form-control" id="firstName" required
      [(ngModel)]="employee.firstName" name="firstName">
    </div>

    <div class="form-group">
      <label for="name">Last Name</label>
      <input type="text" class="form-control" id="lastName" required
      [(ngModel)]="employee.lastName" name="lastName">
    </div>

    <div class="form-group">
      <label for="name">Email Id</label>
      <input type="text" class="form-control" id="emailId" required
      [(ngModel)]="employee.emailId" name="emailId">
    </div>

    <button type="submit" class="btn btn-success">Submit</button>
  </form>
</div>
```

5. Create View Employee Details Template and Component

Here we create view employee details functionality. Let's create an HTML template and component of **Employee** details functionality.

Employee Details Component

Path - `src/app/employee-details/employee-details.component.ts`

The EmployeeDetailsComponent component used to display a particular employee detail. Add the following code to it.

```
import { Employee } from '../employee';
import { Component, OnInit, Input } from '@angular/core';
import { EmployeeService } from '../employee.service';
import { EmployeeListComponent } from '../employee-list/employee-
list.component';
import { Router, ActivatedRoute } from '@angular/router';

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

  id: number;
  employee: Employee;
```

```
constructor(private route: ActivatedRoute,private router: Router,
  private employeeService: EmployeeService) { }
ngOnInit() {
  this.employee = new Employee();

  this.id = this.route.snapshot.params['id'];

  this.employeeService.getEmployee(this.id)
    .subscribe(data => {
      console.log(data)
      this.employee = data;
    }, error => console.log(error));
}

list(){
  this.router.navigate(['employees']);
}
}
```

Employee Details Component Template

Path - src/app/employee-details/employee-details.component.html

The *employee-details.component.html* displays a particular employee detail. Add the following code to it.

```
<h2>Employee Details</h2>

<hr/>
<div *ngIf="employee">
  <div>
    <label><b>First Name: </b></label> {{employee.firstName}}
  </div>
  <div>
    <label><b>Last Name: </b></label> {{employee.lastName}}
  </div>
  <div>
    <label><b>Email Id: </b></label> {{employee.emailId}}
  </div>
</div>

<br>
<br>
<button (click)="list()" class="btn btn-primary">Back to Employee
List</button>
<br>
```

6. Employee Service

Path - src/app/employee.service.ts

The **EmployeeService** will be used to get the data from the backend by calling spring boot APIs. Update the *employee.service.ts* file inside *src/app* directory with the following code to it.

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

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  private baseUrl = 'http://localhost:8080/springboot-crud-rest/api/v1/employees';

  constructor(private http: HttpClient) { }

  getEmployee(id: number): Observable<any> {
    return this.http.get(`${this.baseUrl}/${id}`);
  }

  createEmployee(employee: Object): Observable<Object> {
    return this.http.post(`${this.baseUrl}`, employee);
  }

  updateEmployee(id: number, value: any): Observable<Object> {
    return this.http.put(`${this.baseUrl}/${id}`, value);
  }

  deleteEmployee(id: number): Observable<any> {
    return this.http.delete(`${this.baseUrl}/${id}`, { responseType: 'text' });
  }

  getEmployeesList(): Observable<any> {
    return this.http.get(`${this.baseUrl}`);
  }
}
```

We have completed the development of CRUD operations for an employee model using Angular 8.

Topic 4 - Angular 8 CRUD App Configuration

- npm package.json - Configure Dependencies
- App Routing Module
- App Component
- App Component Template
- App Module
- Main Index Html File
- Main (Bootstrap) File
- Polyfills
- TypeScript tsconfig.json

npm package.json - Configure Dependencies

Path: /package.json

The *package.json* file contains project configuration information including package dependencies which get installed when you run *npm install*. Full documentation is available on the [npm docs website](https://docs.npmjs.com/).

Note that angular version 8.0.0 in the dependencies section in the below file.

```
{
  "name": "angular8-springboot-client",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~8.0.0",
    "@angular/common": "~8.0.0",
    "@angular/compiler": "~8.0.0",
    "@angular/core": "~8.0.0",
    "@angular/forms": "~8.0.0",
    "@angular/platform-browser": "~8.0.0",
    "@angular/platform-browser-dynamic": "~8.0.0",
    "@angular/router": "~8.0.0",
    "bootstrap": "^4.3.1",
    "jquery": "^3.4.1",
    "rxjs": "~6.4.0",
    "tslib": "^1.9.0",
    "zone.js": "^0.9.1"
  },
}
```

```

"devDependencies": {
  "@angular-devkit/build-angular": "~0.800.0",
  "@angular/cli": "~8.0.1",
  "@angular/compiler-cli": "~8.0.0",
  "@angular/language-service": "~8.0.0",
  "@types/node": "~8.9.4",
  "@types/jasmine": "~3.3.8",
  "@types/jasminewd2": "~2.0.3",
  "codemirror": "^5.0.0",
  "jasmine-core": "~3.4.0",
  "jasmine-spec-reporter": "~4.2.1",
  "karma": "~4.1.0",
  "karma-chrome-launcher": "~2.2.0",
  "karma-coverage-istanbul-reporter": "~2.0.1",
  "karma-jasmine": "~2.0.1",
  "karma-jasmine-html-reporter": "^1.4.0",
  "protractor": "~5.4.0",
  "ts-node": "~7.0.0",
  "tslint": "~5.15.0",
  "typescript": "~3.4.3"
}
}

```

App Routing Module

Path: /src/app/app.routing.module.ts

Routing for the Angular app is configured as an array of **Routes**, each component is mapped to a path so the Angular Router knows which component to display based on the URL in the browser address bar.

```

import { EmployeeDetailsComponent } from './employee-details/employee-
details.component';
import { CreateEmployeeComponent } from './create-employee/create-
employee.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EmployeeListComponent } from './employee-list/employee-list.component';
import { UpdateEmployeeComponent } from './update-employee/update-
employee.component';

const routes: Routes = [
  { path: '', redirectTo: 'employee', pathMatch: 'full' },
  { path: 'employees', component: EmployeeListComponent },
  { path: 'add', component: CreateEmployeeComponent },
  { path: 'update/:id', component: UpdateEmployeeComponent },
  { path: 'details/:id', component: EmployeeDetailsComponent },
];

```

```
@NgModule({
```



```

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

```

App Component

Path: /src/app/app.component.ts

The app component is the root component of the application, it defines the root tag of the app as with the selector property of the **@Component** decorator.

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular 8 + Spring Boot 2 CRUD Tutorial';
}

```

App Component Template

Path: /src/app/app.component.html

Defines the HTML template associated with the root *AppComponent*.

```

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a routerLink="employees" class="nav-link"
routerLinkActive="active">Employee List</a>
    </li>
    <li class="nav-item">
      <a routerLink="add" class="nav-link" routerLinkActive="active">Add
Employee</a>
    </li>
  </ul>
</nav>
<div class="container">
  <br>
  <h2 style="text-align: center;">{{title}}</h2>
  <hr>
  <div class="card">

```

```

    <div class="card-body">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<footer class="footer">
  <div class="container">
    <span>All Rights Reserved 2019 @JavaGuides</span>
  </div>
</footer>

```

App Module

Path: /src/app/app.module.ts

Defines the root module, named *AppModule*, that tells Angular how to assemble the application. Initially declares only the *AppComponent*. As you add more components to the app, they must be declared here.

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CreateEmployeeComponent } from './create-employee/create-employee.component';
import { EmployeeDetailsComponent } from './employee-details/employee-details.component';
import { EmployeeListComponent } from './employee-list/employee-list.component';
import { HttpClientModule } from '@angular/common/http';
import { UpdateEmployeeComponent } from './update-employee/update-employee.component';
@NgModule({
  declarations: [
    AppComponent,
    CreateEmployeeComponent,
    EmployeeDetailsComponent,
    EmployeeListComponent,
    UpdateEmployeeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Main Index Html File

Path: /src/index.html

The main *index.html* file is the initial page loaded by the browser that kicks everything off. Webpack bundles all of the javascript files together and injects them into the body of the *index.html* page so the scripts get loaded and executed by the browser.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular8SpringbootClient</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>
```

Main (Bootstrap) File

Path: /src/main.ts

The main file is the entry point used by angular to launch and bootstrap the application.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Polyfills

Path: /src/polyfills.ts

Some features used by Angular 8 are not yet supported natively by all major browsers, polyfills are used to add support for features where necessary so your Angular 8 application works across all major browsers.

```
import 'core-js/features/reflect';  
import 'zone.js/dist/zone';
```

TypeScript tsconfig.json

Path: /tsconfig.json

The tsconfig.json file configures how the TypeScript compiler will convert TypeScript into JavaScript that is understood by the browser. More information is available on the [TypeScript docs](#).

```
{  
  "compileOnSave": false,  
  "compilerOptions": {  
    "baseUrl": "./",  
    "outDir": "./dist/out-tsc",  
    "sourceMap": true,  
    "declaration": false,  
    "module": "esnext",  
    "moduleResolution": "node",  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "importHelpers": true,  
    "target": "es2015",  
    "typeRoots": [  
      "node_modules/@types"  
    ],  
    "lib": [  
      "es2018",  
      "dom"  
    ]  
  }  
}
```

We have completed the Angular 8 web application configuration and app-related components.

Topic 5 - Running Angular 8 CRUD App

Running Angular 8 Client Application

Let's run the above developed Angular App with a command:

```
ng serve
```

By default, the Angular app runs on **4200** port but you can change default port with the following command:

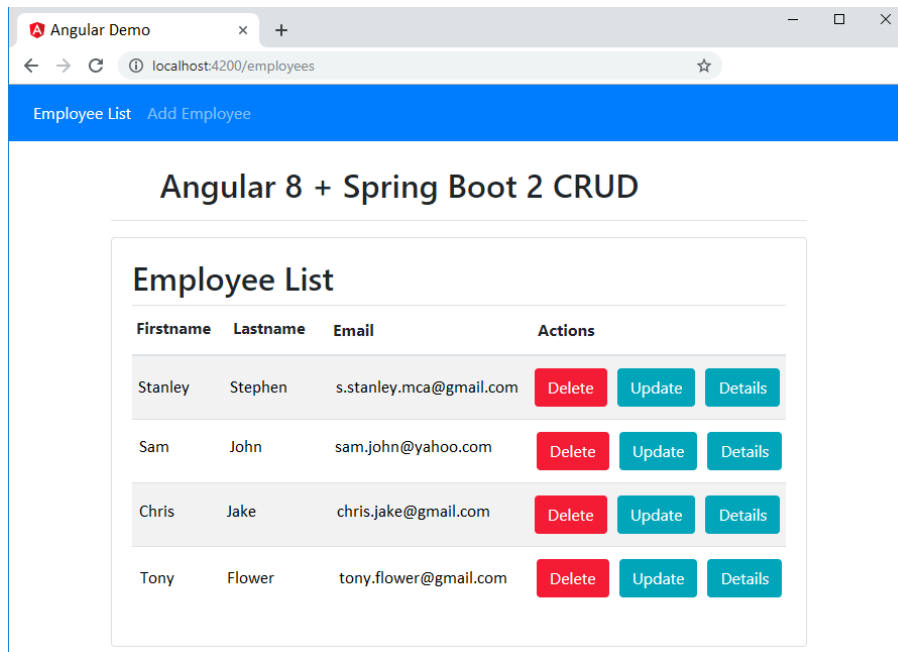
```
ng serve --port 4201
```

Demo

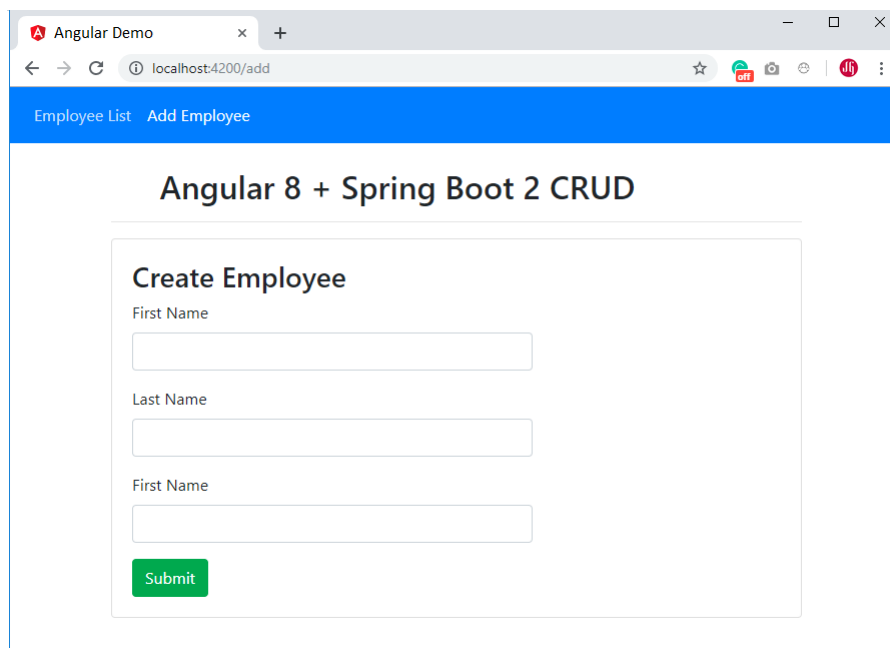
Hit <http://localhost:4200> link in a browser will host this Angular 8 CRUD app.

Below screenshots shows the UI of our **Employee Management System** App:

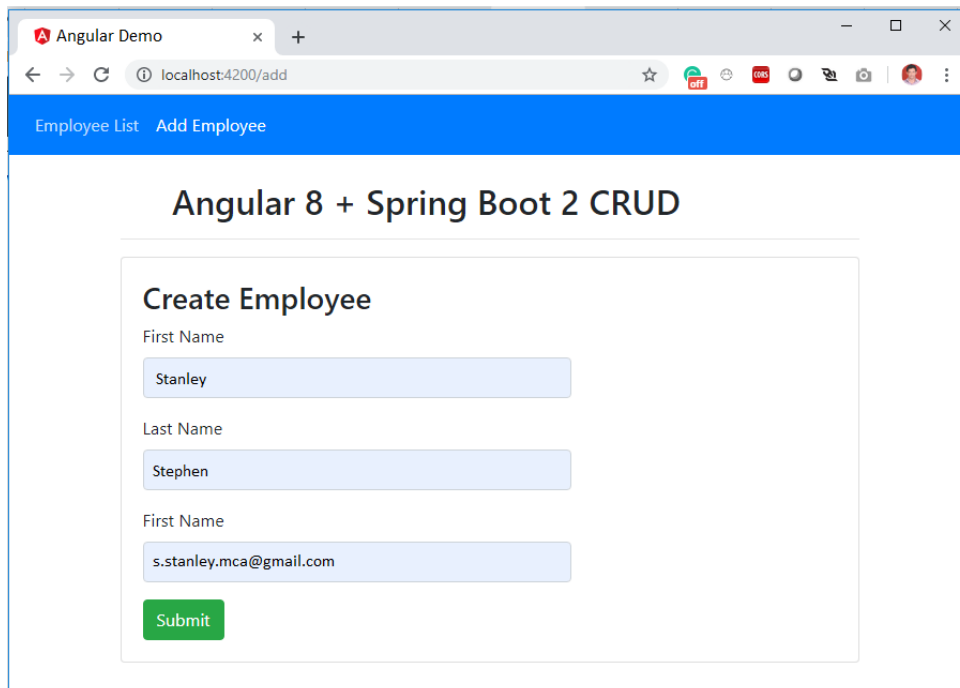
List All Employees Screen



Create Employee Screen

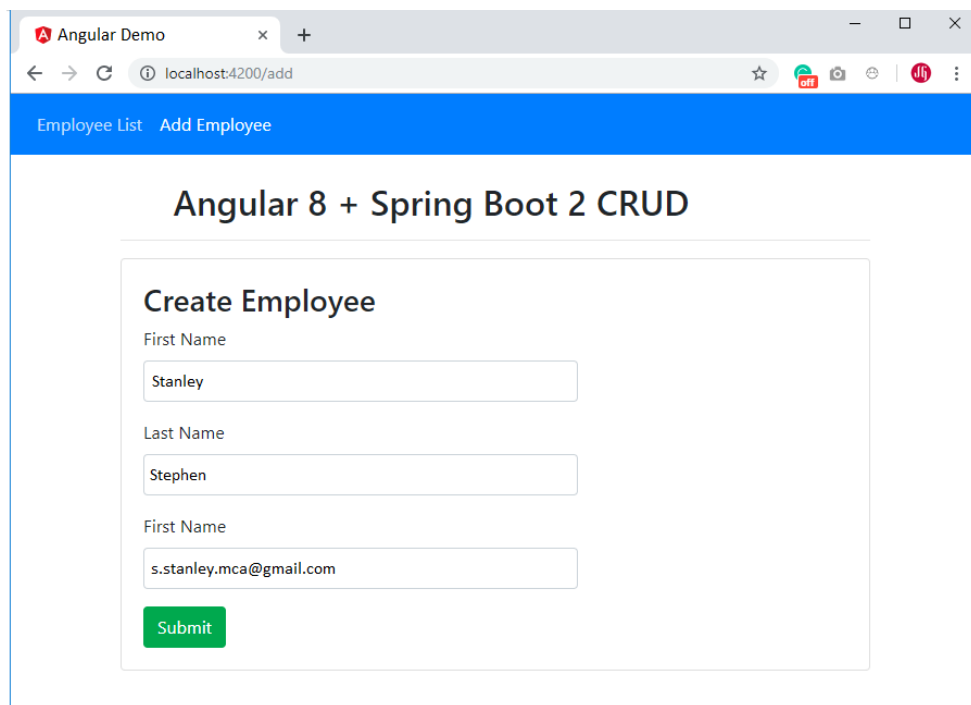


Screen looks as shown below after creating an employee



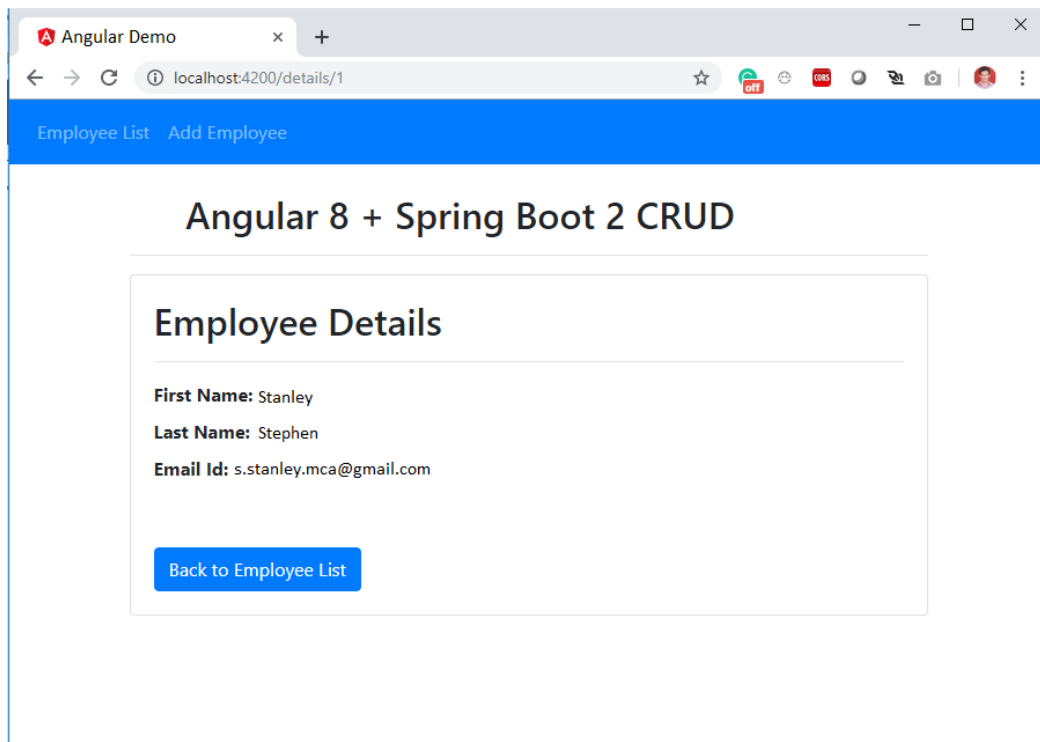
The screenshot shows a web browser window titled 'Angular Demo' with the address bar displaying 'localhost:4200/add'. The browser's navigation bar includes 'Employee List' and 'Add Employee' links. The main content area features the heading 'Angular 8 + Spring Boot 2 CRUD' and a 'Create Employee' form. The form contains three input fields: 'First Name' with the value 'Stanley', 'Last Name' with the value 'Stephen', and 'First Name' with the value 's.stanley.mca@gmail.com'. A green 'Submit' button is located at the bottom of the form.

Update Employee Screen

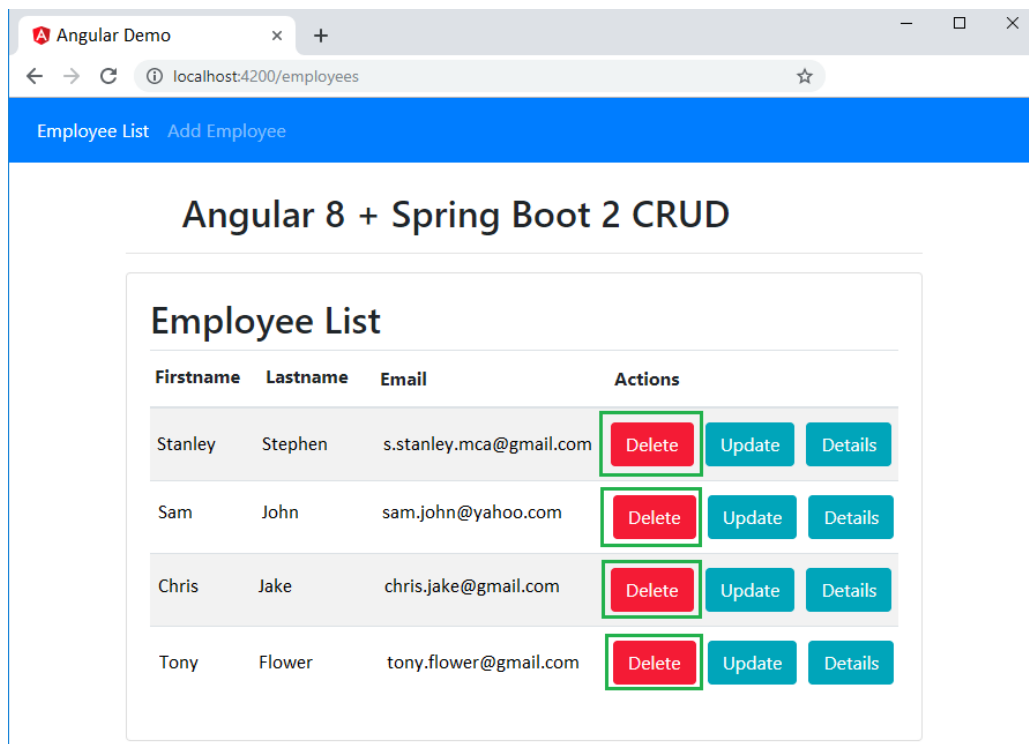


The screenshot shows a web browser window titled 'Angular Demo' with the address bar displaying 'localhost:4200/add'. The browser's navigation bar includes 'Employee List' and 'Add Employee' links. The main content area features the heading 'Angular 8 + Spring Boot 2 CRUD' and a 'Create Employee' form. The form contains three input fields: 'First Name' with the value 'Stanley', 'Last Name' with the value 'Stephen', and 'First Name' with the value 's.stanley.mca@gmail.com'. A green 'Submit' button is located at the bottom of the form.

View Employee Details Screen



Delete Employee Screen



We have completed the development of Angular 8 CRUD application with SpringBoot.