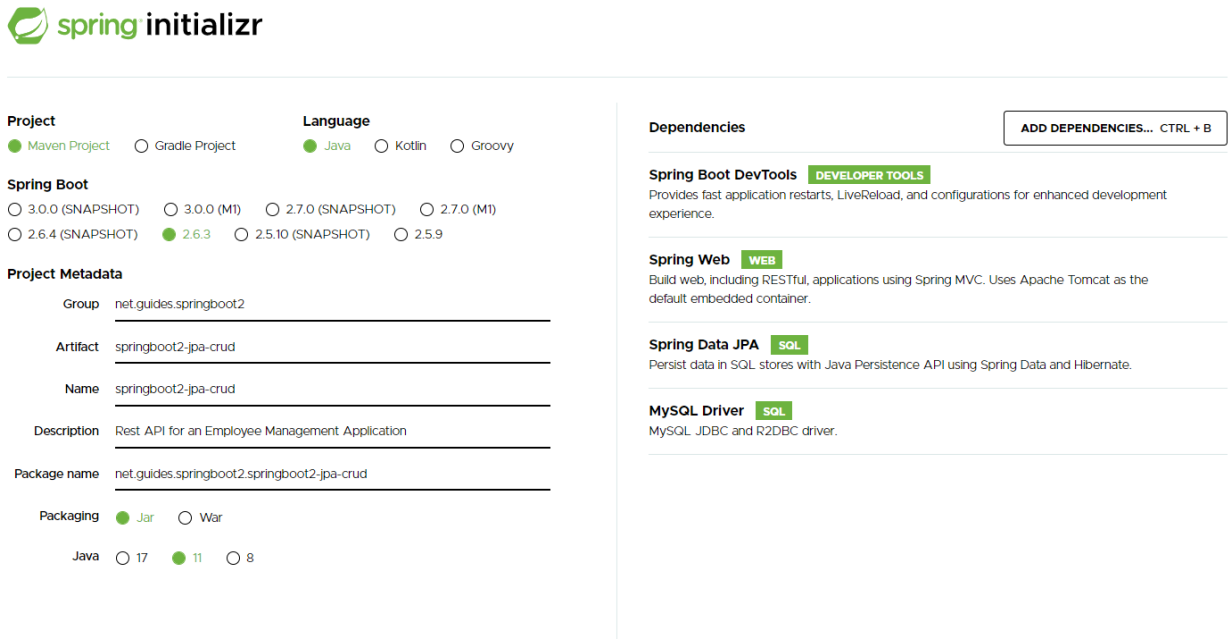
Cloud Innovation Center

Spring Boot & Angular 10 CRUD Tutorial

Spring Boot

1. Begin with creating a Spring Boot application. The easiest way to do so is using a Spring Initializer, which generates a Spring Boot application.



2. The selections for this Spring Boot application are:
   - ※ Project: Maven Project
   - ※ Language: Java
   - ※ Spring Boot: 2.6.3 (or the default selection)
   - ※ Group: net.guides.springboot2
   - ※ Artifact: springboot2-jpa-crud
   - ※ Name: springboot2-jpa-crud
   - ※ Description: Rest API for an Employee Management Application
   - ※ Package Name: net.guides.springboot2.springboot2-jpa-crud

※ Packaging: Jar

※ Java: 11

3. Search for and add the following dependencies:

※ Spring Boot DevTools

※ Spring Web

※ Spring Data JPA

※ MySQL Driver

4. Extract the zipped file into the project folder. This folder should be located in the User directory (NOT in the local Documents directory).
5. In IntelliJ (or your preferred IDE), open the spring boot application folder.
6. Within src\main\java is the package net.guides.springboot2.springboot2.jpacrud. In this package, create the following packages and .java files.

※ controller

i. EmployeeController.java

※ exception

i. ErrorDetails.java

ii. GlobalExceptionHandler.java

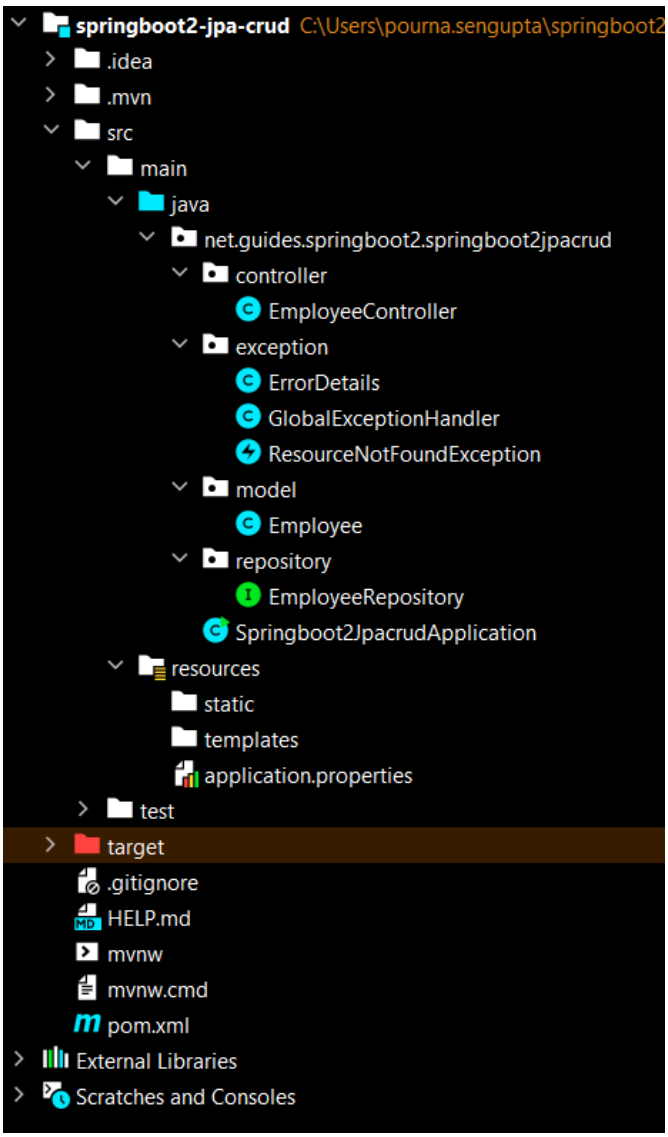
iii. ResourceNotFoundException.java

※ model

i. Employee.java

※ repository

i. EmployeeRepository.java



7. The *pom.xml* file should be created with the correct dependencies, as selected in the Spring Boot generator.
8. Edit the *application.properties* file in the *resources* directory. Check that the localhost, username, and password match the database configuration used in the MySQL local host server.

```
spring.datasource.url = jdbc:mysql://localhost:3306/EMS?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
```

..\springboot2-jpa-crud\src\resources\application.properties

MySQL Database

1. To install and launch MySQL Workbench, please follow this step-by-step guide.
2. Launch the local MySQL server and run the following query

```
create DATABASE ems;
```

Model Layer

9. Within the Model Layer is *Employee.java*. In this file, we will be creating an Employee JPA entity. In this file, the following fields are created.
   - ※ ID: Primary Key
   - ※ firstName: First Name of Employee
   - ※ lastName: Last Name of Employee
   - ※ phoneNum: Phone Number

```java
package net.guides.springboot2.springboot2jpacrud.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 phoneNum;

    public Employee() {

    }

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

    @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 = "phone_number", nullable = false)
    public String getPhoneNum() { return phoneNum; }
    public void setPhoneNum(String phoneNum) { this.phoneNum = phoneNum; }

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

}
```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\model\Employee.java

Repository Layer

10. Within the Repository Layer is *EmployeeRepository.java*, meant to access Employee data from the database. The Spring Data JPA comes with a JpaRepository interface that defines all CRUD operation methods and a default implementation of the JpaRepository.

```java
package net.guides.springboot2.springboot2jpacrud.repository;

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

import net.guides.springboot2.springboot2jpacrud.model.Employee;

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

}

```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\repository\EmployeeRepository.java

Controller Layer

11. Within the Controller Layer, REST APIs for creating, retrieving, updating and deleting an Employee is included in *EmployeeController.java*.

```java
package net.guides.springboot2.springboot2jpacrud.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 org.springframework.web.bind.annotation.CrossOrigin;

import net.guides.springboot2.springboot2jpacrud.exception.ResourceNotFoundException;
import net.guides.springboot2.springboot2jpacrud.model.Employee;
import net.guides.springboot2.springboot2jpacrud.repository.EmployeeRepository;

@CrossOrigin(origins = "http://localhost:4200")
@RestController
@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.setPhoneNum(employeeDetails.getPhoneNum());
        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 {
```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\controller\EmployeeController.java

Exception (Error) Handling

12. Spring Boot provides default implementation for exception handling for RESTful services. Within *ResourceNotFoundException.java,* the following code allows for an a specified Response Status for a specific exception.

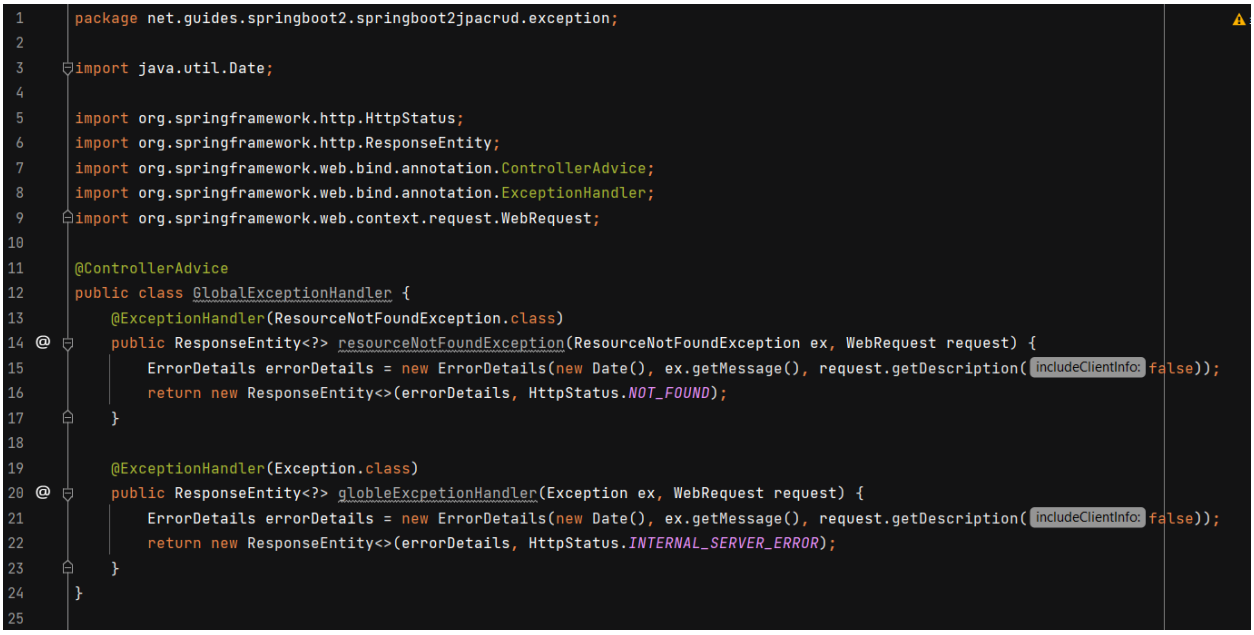
```java
package net.guides.springboot2.springboot2jpacrud.exception;

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);
    }
}

```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\exception\ResourceNotFoundException.java

13. *ErrorDetails.java* holds the code for an error response structure as a simple error response bean. This error response is return through the *GlobalExceptionHandler.java* program handling exception-specific and global exceptions.

```java
package net.guides.springboot2.springboot2jpacrud.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;
    }
}
```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\exception\ErrorDetails.java



```java
package net.guides.springboot2.springboot2jpacrud.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( includeClientInfo: 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( includeClientInfo: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud\exception\GlobalExceptionHandler.java

---

Running the Spring Boot Application

14. The Spring Boot application has an entry point Java class *Springboot2JpacrudApplication.java*. To run the full Spring Boot application, only this file needs to be run. This should be a continuous program, that is ended once the instance has been terminated.

```java
package net.guides.springboot2.springboot2jpacrud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springboot2JpacrudApplication {

    public static void main(String[] args) {
        SpringApplication.run(Springboot2JpacrudApplication.class, args);
    }

}
```

..\springboot2-jpa-crud\src\main\java\net.guides.springboot2.springboot2jpacrud
\Springboot2JpacrudApplication.java

Angular 10 Client

15. Developing the CRUD Web Application is done using Angular 10. Node.js will need to be installed. The download can be found here.
16. Once NPM is available, run the following line in the terminal. This will install Angular CLI 10. The ideal directory is the User directory.

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

17. To create a new Angular 10 Client, run the following command.

```
ng new angular10-client
```

18. If *ng* commands are not understood by the terminal interface, please restart it. If the problem still persists, check to make sure the two following lines are included in the PATH environment variable.

```
C:\Users\pourna.sengupta\AppData\Roaming\npm
```

```
C:\Users\pourna.sengupta\AppData\Roaming\npm\node_modules\@angular\cli
```

19. The terminal command to create a new Angular 10 Client will ask the following questions. Please select the bolded answer.

```
Would you like to add Angular routing? (y/n) YES
```

```
Which stylesheet format would you like to use? CSS
```

20. Once the packages have ran successfully, we can auto-generate the service and components using the Angular CLI. Travel to the directory *angular10-client\src\app* and run the following commands. The ng command s creates new services and the command c creates new components.

```
ng g s employee
```

```
ng g c create-employee
```

```
ng g c employee-details
```

```
ng g c employee-list
```

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

21. Install Bootstrap and jQuery into the node_modules folder.

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

22. Add the following styles and scripts to configure the installed Bootstrap and jQuery in the *angular.json* files.

```
"root": "",
"sourceRoot": "src",
"prefix": "app",
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
      "outputPath": "dist/angular10-client",
      "index": "src/index.html",
      "main": "src/main.ts",
      "polyfills": "src/polyfills.ts",
      "tsConfig": "tsconfig.app.json",
      "assets": [
        "src/favicon.ico",
        "src/assets"
      ],
      "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"
      ]
```

..\angular10-client\anglar.json

Employee

23. Create a new file, *Employee.ts* in the *..\src\app* directory. Include the following code. This defines an Employee class that works with the employees and their details.

```
1   export class Employee {
2       id!: number;
3       firstName!: string;
4       lastName!: string;
5       phoneNum!: string;
6       active!: boolean;
7   }
```

..\angular10-client\src\app\employee.ts

Employee List Component

24. The first component to be created is the Employee List Template and Component. Add the following code to *employee-list.component.ts*. This component displays a list of employees, creates a new employee, and deletes an employee.

```typescript
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]);
  }

  updateEmployee(id: number){
    this.router.navigate(['update', id])
  }
}
```

..\angular10-client\src\app\employee-list\employee-list.component.ts

25. Next, update the *employee-list.component.html* to update the html site hosting the CRUD application. Each of the HTML forms shows the component in the hosted application frontend.

```
<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>First Name</th>
            <th>Last Name</th>
            <th>Phone Number</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let employee of employees | async">
            <td>{{employee.firstName}}</td>
            <td>{{employee.lastName}}</td>
            <td>{{employee.phoneNum}}</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>
                <button (click)="updateEmployee(employee.id)" class="btn btn-info" style="margin: 20px">Update</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>

```

..\angular10-client\src\app\employee-list\employee-list.component.html

Create Employee Component

26. Update *create-employee.component.ts* and *create-employee.component.html* with the following code. This component creates and handles new employee form data.

..\angular10-client\src\app\create-employee \create-employee.component.ts

```ts
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']);
  }
}
```

```html
<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">Phone Number</label>
      <input type="text" class="form-control" id="phoneNum" required [(ngModel)]="employee.phoneNum" 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>
```

..\angular10-client\src\app\create-employee \create-employee.component.html

Update Employee Component

27. Update *update-employee.component.ts* and *update-employee.component.html* with the following code. This component updates an existing employee by getting the object using REST API and populating the HTML form using data binding. Users can edit the employee form data and submit the form again.

..\angular10-client\src\app\update-employee\update-employee.component.ts

```ts
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 = new Employee;
  submitted = false;

  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']);
  }
}
```

..\angular10-client\src\app\update-employee\update-employee.component.html

```html
<h3>Update 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">Phone Number</label>
      <input type="text" class="form-control" id="emailId" required [(ngModel)]="employee.phoneNum" name="emailId">
    </div>

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

Employee Details Component

28. Update *employee-details.component.ts* and *employee-details.component.html* with the following code. In the Employee Details component, a particular employee's details are displayed.

```typescript
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 = new 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']);
  }
}
```

..\angular10-client\src\app\employee-details \employee-details.component.ts

```
<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>Phone Number: </b></label> {{employee.phoneNum}}
  </div>
</div>

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

..\angular10-client\src\app\employee-details \employee-details.component.html

---

Employee Service

29. Update *employee.service.ts* to match the following code. This retrieves the data from the backend by calling Spring Boot APIs.

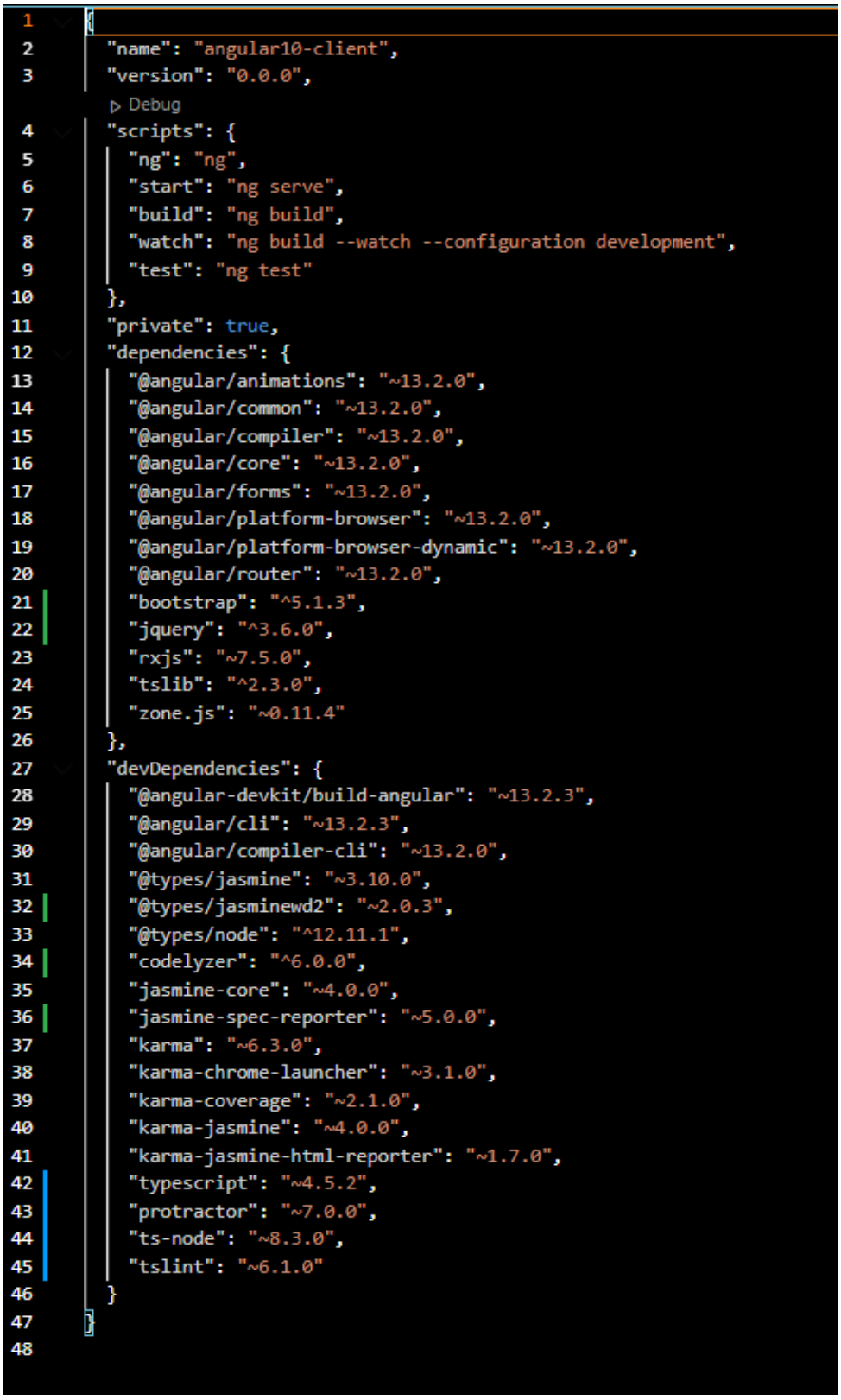
```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@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}`);
  }
}
```

..\angular10-client\src\app\employee.service.ts

Package Dependencies

30. Cross check *package.json* with the following to ensure that all necessary dependencies are included. *Package.json* files contain project configuration information, like package dependencies.

..\angular10-client\package.json



```
{
  "name": "angular10-client",
  "version": "0.0.0",
  ▷ Debug
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~13.2.0",
    "@angular/common": "~13.2.0",
    "@angular/compiler": "~13.2.0",
    "@angular/core": "~13.2.0",
    "@angular/forms": "~13.2.0",
    "@angular/platform-browser": "~13.2.0",
    "@angular/platform-browser-dynamic": "~13.2.0",
    "@angular/router": "~13.2.0",
    "bootstrap": "^5.1.3",
    "jquery": "^3.6.0",
    "rxjs": "~7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~13.2.3",
    "@angular/cli": "~13.2.3",
    "@angular/compiler-cli": "~13.2.0",
    "@types/jasmine": "~3.10.0",
    "@types/jasminewd2": "~2.0.3",
    "@types/node": "^12.11.1",
    "codelyzer": "^6.0.0",
    "jasmine-core": "~4.0.0",
    "jasmine-spec-reporter": "~5.0.0",
    "karma": "~6.3.0",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage": "~2.1.0",
    "karma-jasmine": "~4.0.0",
    "karma-jasmine-html-reporter": "~1.7.0",
    "typescript": "~4.5.2",
    "protractor": "~7.0.0",
    "ts-node": "~8.3.0",
    "tslint": "~6.1.0"
  }
```

Application Routing

31. Routing for Angular applications is configured as arrays of Routes, where each component is mapped to a path, providing the Angular Router with the display for each component base on the URL in the browser address. Update *app-routing.module.ts*.

```
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 { }
```

..\angular10-client\src\app\app-routing.module.ts

Application Root Component

32. The root component of the application is named the app component in this example. This defines the root tag of the application as the selector property of the @Component decorator. To understand this in more detail, please read about wrappers here. Update *app.component.ts* to the following.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular 10 + Spring Boot 2 CRUD';
}
```

..\angular10-client\src\app\app.component.ts

33. Define the HTML template for the root AppComponent in the HTML file. Update *app.component.html* to the following.

```html
<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>2022 @PournaSengupta</span>
  </div>
</footer>
```

..\angular10-client\src\app\app.component.html

34. Define the root module, AppModule by updating *app.component.html* to the following. This communicates with Angular how to assemble the application created. If more components are necessary, they must be declared here to be functional.

```
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 { }
```

..\angular10-client\src\app\app.module.ts

Main Program Files

35. Within the index.html file, the initial page template can be found. Webpack bundles all javascript files together and adds them into the index.html body to load and execute the scripts in the browser.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular10SpringBootClient</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>
```

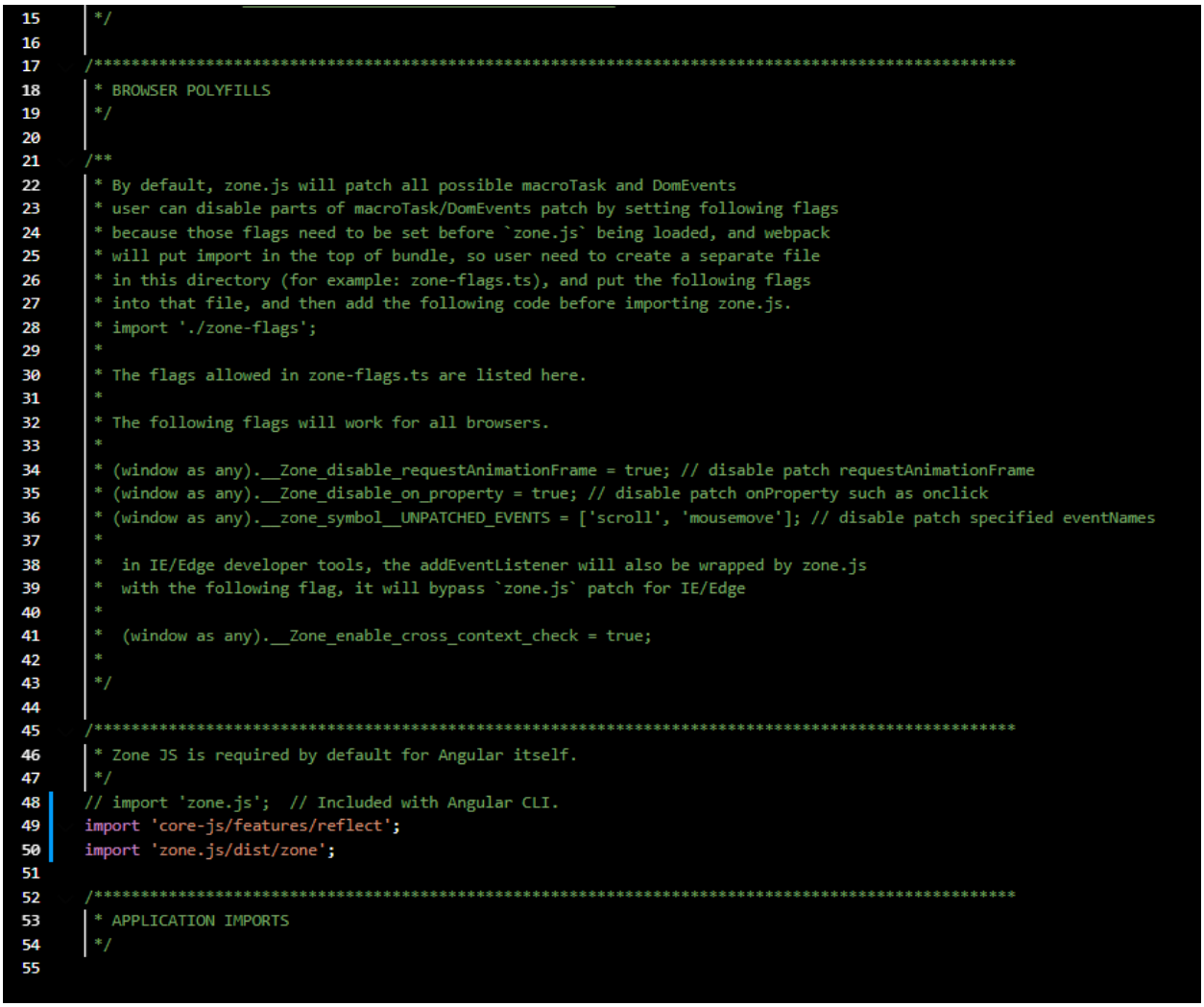
..\angular10-client\src\index.html

36. The main.ts file is an entry point for Angular to launch and bootstrap the CRUD 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));
```

..\angular10-client\src\main.ts

37. Within polyfills.ts, two lines are added, as shown below. Polyfills are added for support of features where needed as not all Angular 10 features are supported by all major browsers.



```
 */

/***************************************************************************************************
 * BROWSER POLYFILLS
 */

/**
 * By default, zone.js will patch all possible macroTask and DomEvents
 * user can disable parts of macroTask/DomEvents patch by setting following flags
 * because those flags need to be set before `zone.js` being loaded, and webpack
 * will put import in the top of bundle, so user need to create a separate file
 * in this directory (for example: zone-flags.ts), and put the following flags
 * into that file, and then add the following code before importing zone.js.
 * import './zone-flags';
 *
 * The flags allowed in zone-flags.ts are listed here.
 *
 * The following flags will work for all browsers.
 *
 * (window as any).__Zone_disable_requestAnimationFrame = true; // disable patch requestAnimationFrame
 * (window as any).__Zone_disable_on_property = true; // disable patch onProperty such as onclick
 * (window as any).__zone_symbol__UNPATCHED_EVENTS = ['scroll', 'mousemove']; // disable patch specified eventNames
 *
 *  in IE/Edge developer tools, the addEventListener will also be wrapped by zone.js
 *  with the following flag, it will bypass `zone.js` patch for IE/Edge
 *
 *  (window as any).__Zone_enable_cross_context_check = true;
 *
 */

/***************************************************************************************************
 * Zone JS is required by default for Angular itself.
 */
// import 'zone.js';  // Included with Angular CLI.
import 'core-js/features/reflect';
import 'zone.js/dist/zone';

/***************************************************************************************************
 * APPLICATION IMPORTS
 */

```

..\angular10-client\src\polyfill.ts

Running the Full Application

38. To run and host the Angular 10 Client, enter the following command into the terminal.

```
ng serve
```

39. Run *Springboot2JpacrudApplication.java* to start the application backend. Navigate to http://localhost:4200/ to view the Angular 10 CRUD app.
