Curso de Angular Developer

Guia Practica 3A Table Empleados

|  |  |  |  |
| --- | --- | --- | --- |
|  | **NOMBRE Y FUNCIÓN** | **FECHA** | **FIRMA** |
| Preparado por | **Joe Valdivia Lira**  Instructor | 07-03-2018 |  |
| Verificado por | **Edwin Maravi**  Jefe de Proyectos CJava |  |  |
| Aprobado por |  |  |  |
| Autorizado por |  |  |  |

|  |
| --- |
| **Distribución:** |
| Documentación |

**Log de Control de Cambios**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tema** | **Fecha** | **Modificación** | **Observaciones** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Distribución:** |
| Documentación |

Desarrollo de Aplicaciones Web con Angular 6/7

Operaciones CRUD en Angular

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Índice**

1. Acceso de API REST de Spring Boot
   1. Tecnologias
   2. Aplicación Angular
   3. Spring Boot Server
   4. Cliente Angular
2. Creacion del Proyecto Spring Boot
   1. Crear el proyecto
   2. Importacion del proyecto con Spting Boot
   3. Dependencias
   4. Modelo de datos
   5. JPA Repsoitory

1 **Instalacion de Angular Material**

### 1.1 Introduccion

En este tutorial aprenderemos como desarrollar un single page application (SPA) usando Angular 6 como front-end y Spring boot restful API as backend.

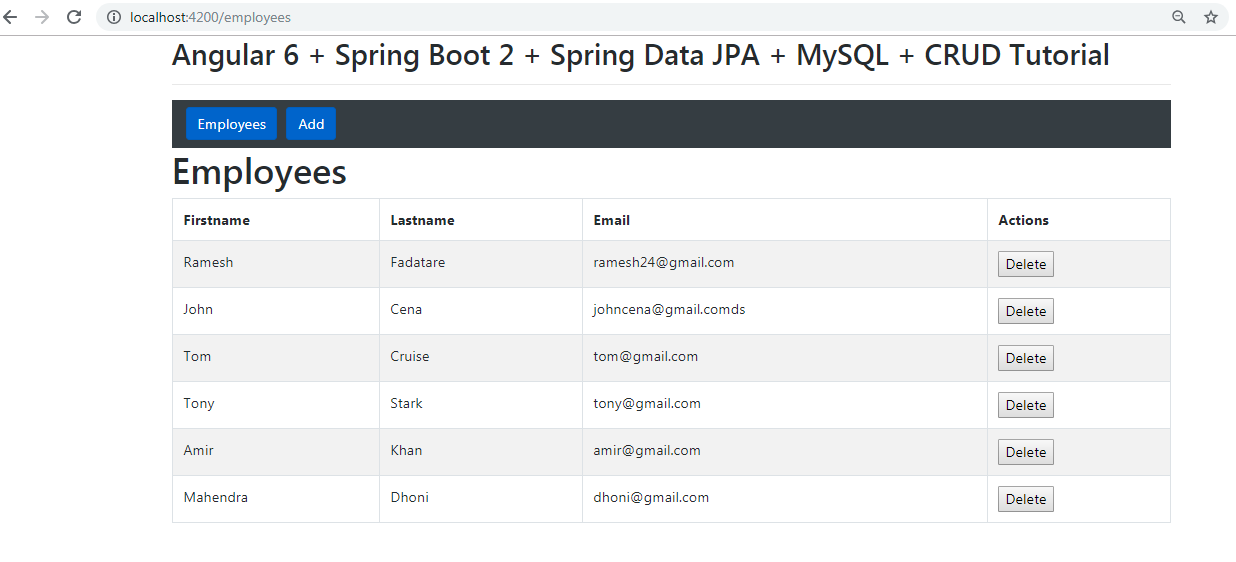
**Autor original by**[**Ramesh Fadatare**](https://www.blogger.com/profile/14691512106162803120)**on [February 03, 2019](https://www.javaguides.net/2019/02/spring-boot-angular-6-crud-example.html" \o "permanent link)**

**Modifiaciones: Joe Valdivia Lira en Marzo 07, 2019**

## **Que construiremos ?**

Basicamente se crearan dos proyectos:

1. **springboot2-jpa-crud-example**: Este proyecto es usado para desarrollar CRUD RESTFul APIs para un simple **Employee Management System** usando Spring Boot 2, JPA y MySQL como una database.
2. **angular6-springboot-client**: Este proyecto es usado para desarrollar un single page application usando Angular 6 como front-end technology. Esta aplicación angular 6 consume CRUD Restful APIs desarrollado por un  **springboot2-jpa-crud-example** project.

**[](https://4.bp.blogspot.com/-IfDE7KWC8Vg/XFbmFM3BFLI/AAAAAAAAFhE/MaIJbeY-axI7iMqLCL7LTH3Iuasa5pErQCLcBGAs/s1600/employee-list.PNG)**

## Herramientas y tecnologías usadas

### Server-side technologies

* Spring Boot - 2.0.4.RELEASE
* JDK - 1.8 or later
* Spring Framework - 5.0.8 RELEASE
* Hibernate - 5.2.17.Final
* Spring Data JPA - 2+

### Front end technologies

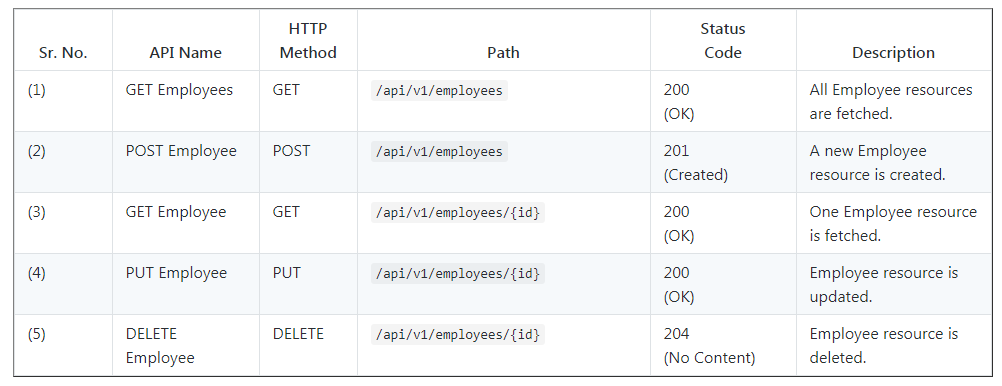
* Angular 6/7
* Bootstrap 4
* npm- 6.4.1

### Tools

* Maven - 3.2+
* IDE - Eclipse or Spring Tool Suite (STS)
* Visual Studio 2017
* Angular CLI

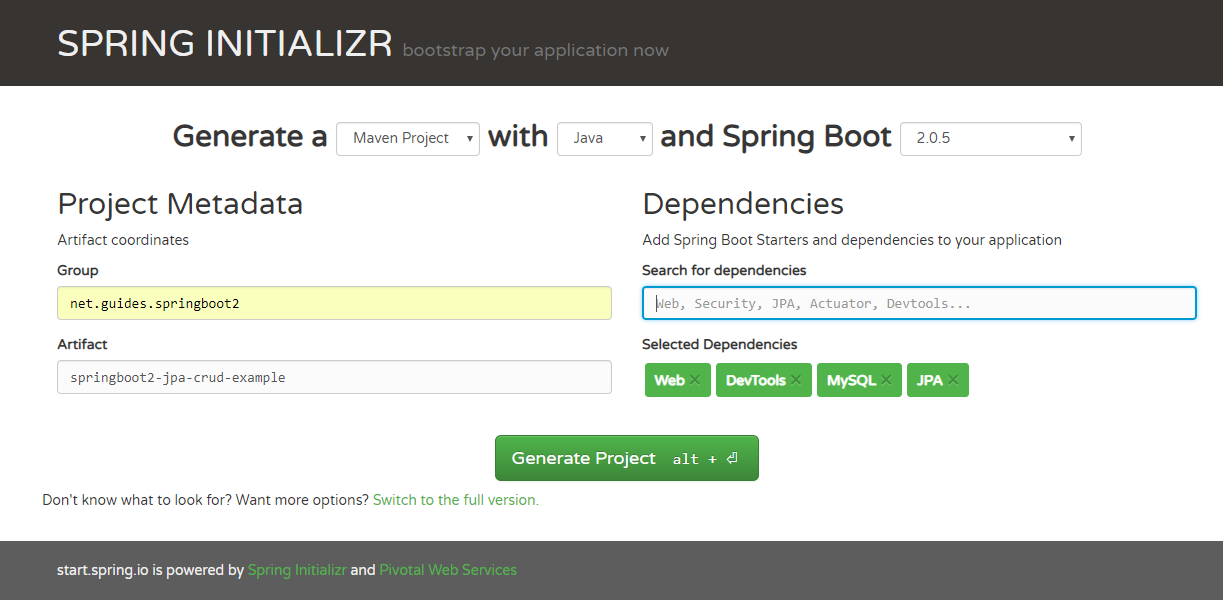
### 1.2 Creando el Spring Boot CRUD Rest APIs

1. Primeramente construiremos un **CRUD RESTFul APIs** usando Spring Boot 2 JPA and MySQL. Mas tarde consumiremos estos Rest APIs usando una aplicaicon Angular 6 Tenemos cinco REST APIs (métodos en Controller) creados para el recurso  Employee.

**[](https://4.bp.blogspot.com/-GIlhRU_3AdQ/W5trMB5AyII/AAAAAAAADyE/b8LXiR5fiFYC2Z9GdYiOYjV6w5gZv9jaQCLcBGAs/s1600/rest-api-list.PNG)**

### 1.2 Creando e Importando el Project

Existen muchas formas de crear una aplicacion Spring Boot. La forma mas simple es usar Spring Initializr en [**http://start.spring.io/**](http://start.spring.io/), el cual es un generador de aplicaciones Spring Boot en linea.

**[](https://2.bp.blogspot.com/-1PSt3-IwbAE/W5tr3IUzS8I/AAAAAAAADyU/pgT0-em1wDo6ai2fTJ6NCGofOSU7VJ5yQCLcBGAs/s1600/create-project.PNG)**

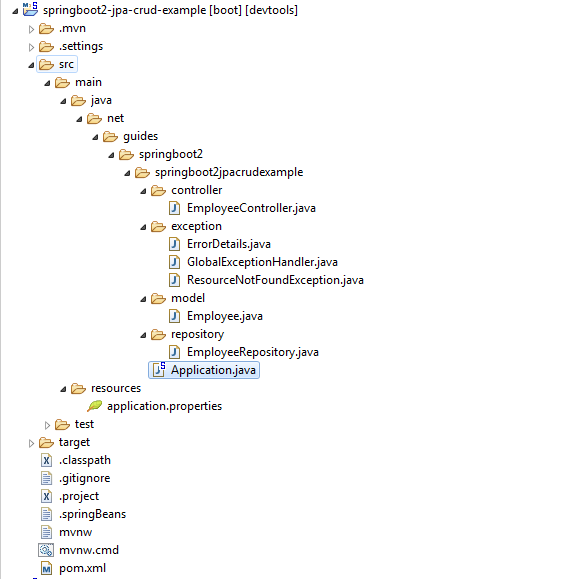
Nosotros especificamos los siguientes detalles:

* **Generate**: Maven Project
* **Java Version**: 1.8 (Default)
* **Spring Boot**: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

Una vez , todos los detalles son ingresados, damos click sobre el botón Generate Project para que se genere el spring boot project and downloads. A continuación desempaquetamos (Unzip) el archivo descargado y lo importamos al STS.

### 1.3 Estructura de paquetes

Siguiendo la estructura de empaquetamiento de nuestro Employee Management System -

**[](https://1.bp.blogspot.com/--r6LOKhuo1c/W5traXxjc5I/AAAAAAAADyQ/3Nttx0BtXF4toSUyAL5MbDgw14xGVXo1gCEwYBhgL/s1600/packaging-structure.PNG)**

### El archivo pom.xml

### Verificamos la correcta formación del archivo pom.xml

<?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>mysql</groupId>

<artifactId>mysql-connector-java</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>

### 1.5 Creacion del 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

+ "]";

}

}

### 1.6 Creacion del 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>{

}

### 1.7 Creacion del 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

@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;

}

}

### 1.8 Corriendo la Application

Esta aplicacion spring boot tiene una clase Java como un punto de entrada llamada SpringBootCrudRestApplication.java con el  public static void main(String[] args) metodo, el cual puede ejecutar o correr la aplicacion.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

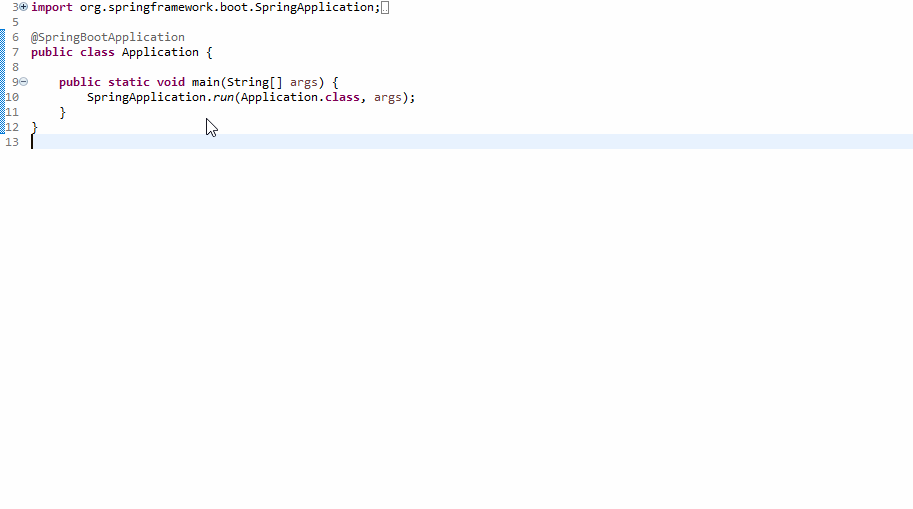
public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The main() method uses Spring Boot’s SpringApplication.run() method to launch an application. You can refer below gif file:

**[](https://1.bp.blogspot.com/-19si4SQ9vJo/XJ3to8ycnwI/AAAAAAAAFzM/L8fwPZMU4D4nWy9_R70UEwXf1M03ED24gCLcBGAs/s1600/running-spring-boot-application.gif)**

Podemos arrancar la aplicación via línea de comandos usando **mvn spring-boot:run**.  
  
This completes the development of Spring boot CRUD Rest APIs. Now we will develop client application using Angular 6.

### 1.9 Aplicación Cliente Angular 6/7

Desarrollamos un Angular 6 SPA (single page application) para consumir las API REST.

ng new angular6-springboot-client

### 1.10 Components, Services y Modules

Listamos los componentes que debemos crear: components, service y modules.

Usaremos [**Angular CLI**](https://angular.io/cli) para generarlos.

1. **Components**

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

1. **Services**

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

1. **Modules**

* FormsModule
* HttpClientModule
* AppRoutingModule.

1. **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.

### Creando Servicios y Componentes

Los generamos usando [**Angular CLI**](https://angular.io/cli) . Verificar de trabajar siempre en el directorio generado para el proyecto: *angular6-springboot-client\src\app* y correr los siguientes comandos:

- ng g s employee

– ng g c create-employee

– ng g c employee-details

– ng g c employee-list

Lista completa de comandos:

C:\angular6\angular6-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:\angular6\angular6-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:\angular6\angular6-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 (629 bytes)

C:\angular6\angular6-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 (737 bytes)

Nosotros usaremos bootstrap 4 para estilos en nuestra aplicacion.

### Integrando Bootstrap con Angular

Usamos NPM para descargar Bootstrap & JQuery. Bootstrap y jQuery serán instalados dentro del folder node\_modules.

npm install bootstrap jquery --save

Configuramos Bootstrap y JQuery en el archivo *angular.json*:

...

"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"

]

...

### Creamos la Clase Employee employee.ts

Antes de definir el *EmployeeListComponent*, definimos una clase *Employee* para trabajar con los empleados. Creamos un archivo *employee.ts* dentro del folser *src/app* y adicionamos el siguiente codigo -

export class Employee {

id: number;

firstName: string;

lastName: string;

emailId: string;

active: boolean;

}

### EmployeeService - employee-list.component.ts

El *EmployeeService* será usado para obtener datos desde el backend llamando spring boot APIs. Actualizamos el archivo *employee.service.ts* dentro del directorio  *src/app* con el siguiente código.

import { Injectable } from '@angular/core';

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

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

private baseUrl = '/api/v1/employees';

constructor(private http: HttpClient) { }

getEmployee(id: number): Observable<Object> {

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}`);

}

}

### EmployeeListComponent - employee-list.component.ts

Actualizamos el componentes *EmployeeListComponent* el cual será usado para mostrar una lista de empleados, crear un nuevo empeado y borrar otros.

Actualizamos/borramos el contenido de *todo-list.component.ts* dentro del directorio *src/app* y adicionamos lo siguiente

import { Observable } from "rxjs";

import { EmployeeService } from "./../employee.service";

import { Employee } from "./../employee";

import { Component, OnInit } from "@angular/core";

@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) {}

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));

}

}

### Create a template for EmployeeListComponent employee-list.component.html

Actualizamos *employee-list.component.html* con el siguiente código.

<div class="panel panel-default">

<div class="panel-heading">

<h1>Employees</h1>

</div>

<div class="panel-body">

<table class="table table-striped table-bordered">

<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)">Delete</button></td>

</tr>

</tbody>

</table>

</div>

</div>

### CreateEmployeeComponent - create-employee.component.ts

*CreateEmployeeComponent* es usado para crear y gestionar un nueo formulario de empleados, adicionamos el siguiente código.

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

import { Employee } from './../employee';

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

@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) { }

ngOnInit() {

}

newEmployee(): void {

this.submitted = false;

this.employee = new Employee();

}

save() {

this.employeeService.createEmployee(this.employee)

.subscribe(data => console.log(data), error => console.log(error));

this.employee = new Employee();

}

onSubmit() {

this.submitted = true;

this.save();

}

}

### Create a template for EmployeeCreateComponent create-employee.component.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">First Name</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>

</div>

### EmployeeDetailsComponent- employee-details.component.ts

Este componente muestra los detalles de los empleados.

import { Employee } from './../employee';

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

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

import { EmployeeListComponent } from '../employee-list/employee-list.component';

@Component({

selector: 'app-employee-details',

templateUrl: './employee-details.component.html',

styleUrls: ['./employee-details.component.css']

})

export class EmployeeDetailsComponent implements OnInit {

@Input() employee: Employee;

constructor(private employeeService: EmployeeService, private listComponent: EmployeeListComponent) { }

ngOnInit() {

}

}

### Create a template for EmployeeDetailsComponent employee-details.component.html

<div \*ngIf="employee">

<div>

<label>Name: </label> {{employee.firstName}}

</div>

<div>

<label>Age: </label> {{employee.lastName}}

</div>

<div>

<label>Active: </label> {{employee.emailId}}

</div>

<div>

<label>Active: </label> {{employee.active}}

</div>

<span class="button is-small btn-primary" \*ngIf='employee.active' (click)='updateActive(false)'>Inactive</span>

<span class="button is-small btn-primary" \*ngIf='!employee.active' (click)='updateActive(true)'>Active</span>

<span class="button is-small btn-danger" (click)='deleteEmployee()'>Delete</span>

<hr/>

</div>

### AppRoutingModule - 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';

const routes: Routes = [

{ path: '', redirectTo: 'employee', pathMatch: 'full' },

{ path: 'employees', component: EmployeeListComponent },

{ path: 'add', component: CreateEmployeeComponent },

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

### AppComponent - app/app.component.ts

Definimos la lógica para el componente raíz llamado *AppComponent*. La vista asociada con este componente se convierte en la raíz de la jerarquiea de vistas.

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 6 + Spring Boot 2 + Spring Data JPA + MySQL + CRUD Tutorial';

}

### app/app.component.html

Definimos el template HTML asociado con el root *AppComponent*.

<div class="container">

<h2>{{title}}</h2>

<hr>

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<!-- Links -->

<ul class="navbar-nav">

<li class="nav-item">

<a routerLink="employees" class="btn btn-primary active" role="button" routerLinkActive="active">Employees</a>

</li>

<li class="nav-item" style="margin-left: 10px;">

<a routerLink="add" class="btn btn-primary active" role="button" routerLinkActive="active">Add</a>

</li>

</ul>

</nav>

<router-outlet></router-outlet>

</div>

## app/app.module.ts

Definimos el root module, llamado *AppModule*, que le dice a Angular como ensamblar la aplicacion. Inicialmente declaramos únicamente el *AppComponent*. Tanto como necesitemos mas componentes en la app, ellos deben ser declarados aqui.

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';

@NgModule({

declarations: [

AppComponent,

CreateEmployeeComponent,

EmployeeDetailsComponent,

EmployeeListComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

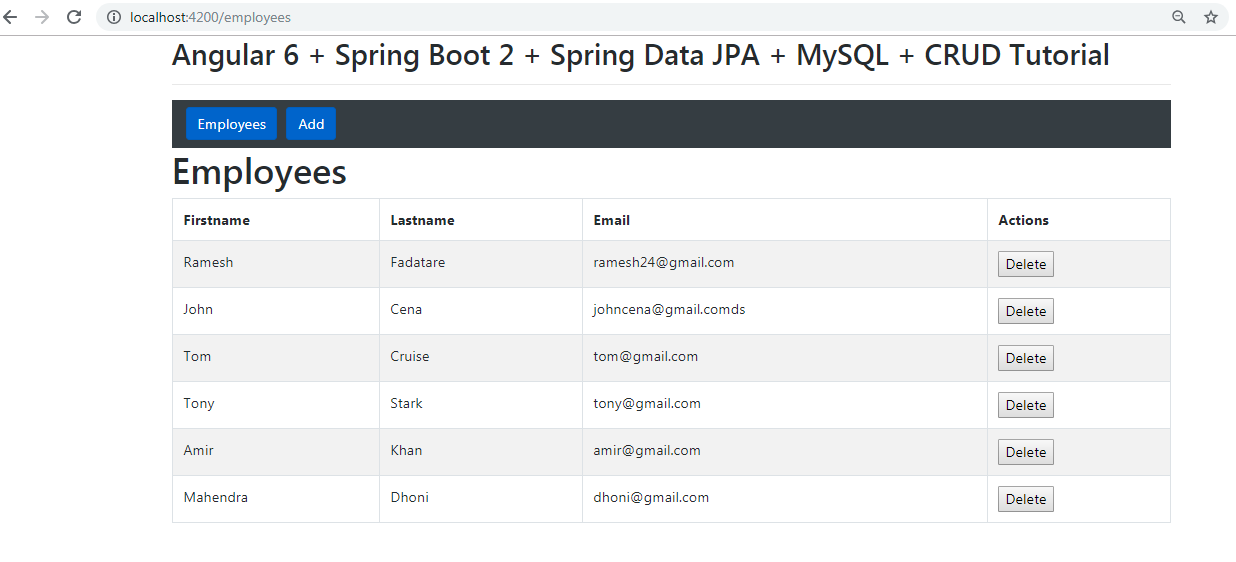
### Running Angular

Ejecutamos el proyecto con el comando: **ng serve**

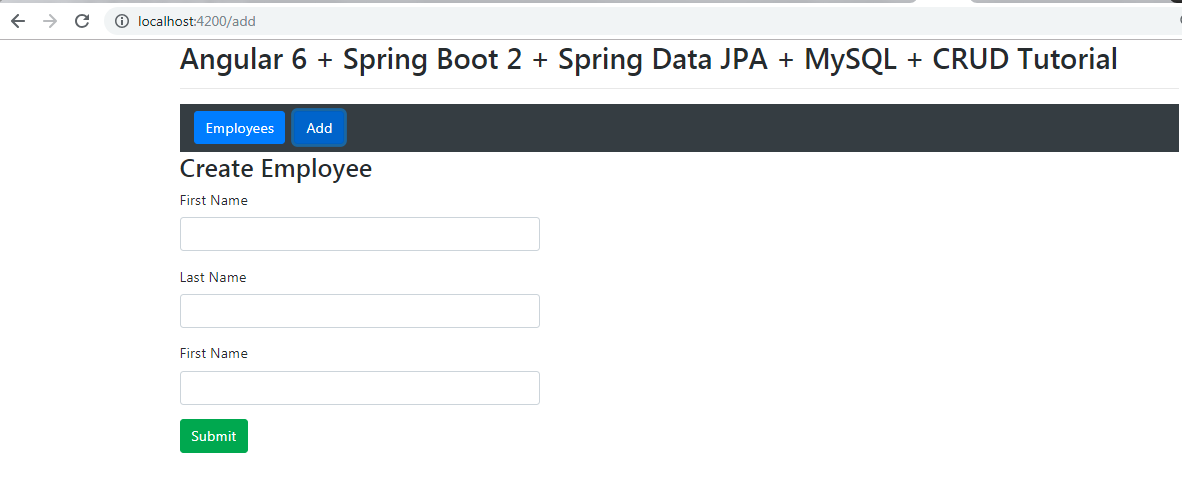
### Salida

Abrimos el browser [**http://URLalhost:4200/**](http://localhost:4200/):

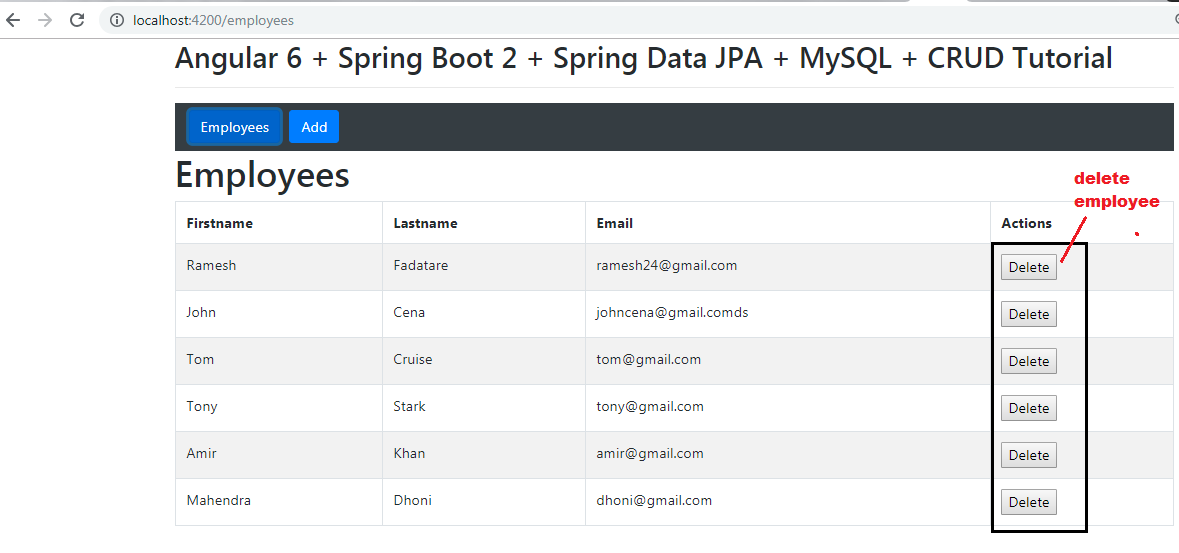
### Employee List Page

**[](https://4.bp.blogspot.com/-IfDE7KWC8Vg/XFbmFM3BFLI/AAAAAAAAFhE/MaIJbeY-axI7iMqLCL7LTH3Iuasa5pErQCLcBGAs/s1600/employee-list.PNG)**

### Add Employee Page

**[](https://4.bp.blogspot.com/-bgSEaNwR6ag/XFbpdvPo6aI/AAAAAAAAFhQ/JljLqd8RwJg4-rvCxbL4QzAyIDJeVnE4QCLcBGAs/s1600/create-employee.PNG)**

## Delete Employee

**[](https://3.bp.blogspot.com/-rPOARDdsyDQ/XFbpnY1xe8I/AAAAAAAAFhU/SODINI0YYsAjCFvGAdnJ-QsNy26t0jKdQCLcBGAs/s1600/delete-employee.PNG)**

La actualización de actualizacon no es implementada, trate de implementarla usted misom (Tip: Adicione un botón update button a la apgina employee list page y que se base en el employee id. Note que el Rest API es actualmente creado para la funcionalidad de actualizacion.  
  
Codigo fuente original en :

[**https://github.com/RameshMF/angular6-springboot-crud-tutorial**](https://github.com/RameshMF/angular6-springboot-crud-tutorial)