Spring Boot Full Stack with Angular

Gustavo García

5 de agosto de 2022.

# El tutorial

<https://youtu.be/Gx4iBLKLVHk>

Muy bueno. Mas de dos horas y media. Es como para un repaso después de haber aprendido cada una de las partes por separado.

Vamos a seguir este tutorial, aunque el proyecto tiene una serie de cambios, a mi gusto. Por ejemplo, cambié los nombres de los paquetes a los que usamos nosotros normalmente. Hay abundantes comentarios en el código. Por un lado ayudan a entender el tutorial, y por otro son material de estudio, de modo que hay que leerlos con cuidado, seguir los links a la documentación, y aprenderlos como para ser evaluados.

C:\Users\Gustavo\My Drive\GIT\30-backend-frontend-amigoscode

## Backend en Spring Boot

La primera parte está dedicada a la construcción del backend, en Spring Boot.

## Frontend en Angular

La segunda parte está dedicada a la construcción del frontend en Angular.

## Integración

Como el frontend accede a la API del backend, hay que integrarlos. Como están en puertos distintos, hay que resolver las restricciones de CORS.

# Backend en Spring Boot

## Servidor de base de datos

Esta aplicación se conecta a una base de datos, de modo que necesitamos que el motor esté corriendo. Yo lo hago con WampServer. Parece que WampServer está armado para MySQL 8.

<https://sourceforge.net/projects/wampserver>

## Crear la base de datos y el usuario

DROP DATABASE IF EXISTS `employeemanager`;

CREATE DATABASE IF NOT EXISTS `employeemanager` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci;

DROP USER IF EXISTS `amigoscode`;

CREATE USER IF NOT EXISTS 'amigoscode' @'%' IDENTIFIED WITH mysql\_native\_password BY 'codeamigos';

GRANT USAGE ON \*.\* TO 'amigoscode' @'%';

GRANT ALL PRIVILEGES ON `employeemanager`.\* TO 'amigoscode' @'%';

USE `employeemanager`;

DROP TABLE IF EXISTS `employee`;

CREATE TABLE IF NOT EXISTS `employee` (

  `id` bigint NOT NULL,

  `email` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci DEFAULT NULL,

  `employee\_code` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci NOT NULL,

  `img\_url` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci DEFAULT NULL,

  `job\_title` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci DEFAULT NULL,

  `name` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci DEFAULT NULL,

  `phone` varchar(255) COLLATE utf8mb4\_es\_0900\_ai\_ci DEFAULT NULL,

  PRIMARY KEY (`id`)

) ENGINE = MyISAM DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_es\_0900\_ai\_ci;

INSERT INTO `employee` (

    `id`,

    `email`,

    `employee\_code`,

    `img\_url`,

    `job\_title`,

    `name`,

    `phone`

  )

VALUES (

    1,

    'uno@gmail.com',

    'bf36bcb2-d0cf-4ce6-b57d-9143ebec4398',

    NULL,

    'UI/UX',

    'Uno',

    NULL

  ),

  (

    2,

    'dos@gmail.com',

    '33943d26-0dff-48ea-b5bd-fac64073b685',

    NULL,

    'UI/UX',

    'Dos',

    NULL

  ),

  (

    3,

    'tres@gmail.com',

    '52307d41-7329-4e70-bfe6-4620868c2ff6',

    NULL,

    'UI/UX',

    'Tres',

    NULL

  ),

  (

    4,

    'cuatro@gmail.com',

    '38ead69b-0194-432c-b96d-6bdd763ea949',

    NULL,

    'UI/UX',

    'Cuatro',

    NULL

  );

## Crear el proyecto

[start.spring.io](https://start.spring.io/)

Maven

Java

2.6.4

com.logicactive

employeemanager

Employee Manager App

Jar

11

Spring Web

Spring Data JPA

MySQL Driver

## src\main\resources\application.properties

# MySQL configuration

spring.datasource.url=jdbc:mysql://localhost:3306/employeemanager

spring.datasource.username=amigoscode

spring.datasource.password=codeamigos

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

## src\main\java\tup\employeemanager\EmployeemanagerApplication.java

/\*

 \* Esta es la clase principal de la aplicación.

 \* Está tal como la genera la plataforma.

 \*/

package tup.employeemanager;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*

  \* La documentación oficial está en

  \* https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/autoconfigure/SpringBootApplication.html

  \* y en

  \* https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html

  \*

  \* La anotación @SpringBootApplication es equivalente a

  \* usar @Configuration, @EnableAutoConfiguration y @ComponentScan

  \* con sus atributos predeterminados.

  \*/

@SpringBootApplication

public class EmployeemanagerApplication {

  public static void main(String[] args) {

    SpringApplication.run(EmployeemanagerApplication.class, args);

  }

}

## src\main\java\tup\employeemanager\controllers\EmployeeController.java

package tup.employeemanager.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

// import org.springframework.web.bind.annotation.CrossOrigin;

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 tup.employeemanager.model.Employee;

import tup.employeemanager.service.EmployeeService;

/\*

 \* Documentación oficial:

 \* https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html

 \*

 \* Una anotación de conveniencia que a su vez está anotada

 \* con @Controller y @ResponseBody. Los tipos que llevan esta

 \* anotación se tratan como controladores donde los

 \* métodos @RequestMapping asumen la semántica @ResponseBody

 \* de manera predeterminada.

 \*

 \* El siguiente tutorial de GeeksforGeeks

 \* https://www.geeksforgeeks.org/spring-rest-controller/

 \* explica qué es y cómo se usa, paso a paso.

 \*/

@RestController

/\*

 \* Esta es la manera de habilitar el CORS para toda la clase.

 \* Hay un modo más complicado, con un bean. Por ahora, este

 \* es el modo que prefiero.

 \*/

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

@RequestMapping("/employee")

public class EmployeeController {

    @Autowired

    private final EmployeeService employeeService;

    public EmployeeController(EmployeeService employeeService) {

        this.employeeService = employeeService;

    }

    @GetMapping("/all")

    public ResponseEntity<List<Employee>> getAllEmployees() {

        List<Employee> employees = employeeService.findAllEmployees();

        return new ResponseEntity<List<Employee>>(employees, HttpStatus.OK);

    }

    @GetMapping("/find/{id}")

    public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") Long id) {

        Employee employee = employeeService.findEmployeeById(id);

        return new ResponseEntity<>(employee, HttpStatus.OK);

    }

    @PostMapping("/add")

    public ResponseEntity<Employee> addEmployee(@RequestBody Employee empl) {

        Employee employee = employeeService.addEmployee(empl);

        return new ResponseEntity<>(employee, HttpStatus.CREATED);

    }

    @PutMapping("/update")

    public ResponseEntity<Employee> updateEmployee(@RequestBody Employee empl) {

        Employee employee = employeeService.updateEmployee(empl);

        return new ResponseEntity<>(employee, HttpStatus.CREATED);

    }

    @DeleteMapping("/delete/{id}")

    public ResponseEntity<?> deleteEmployee(@PathVariable("id") Long id) {

        employeeService.deleteEmployee(id);

        return new ResponseEntity<>(HttpStatus.OK);

    }

}

{

    "email":"uno@gmail.com",

    "imageUrl":"https://foto.com/uno",

    "jobTitle":"UI/UX",

    "name":"Uno",

    "pjone":"123 456 789"

}

{

    "email":"dos@gmail.com",

    "imageUrl":"https://foto.com/dos",

    "jobTitle":"UI/UX",

    "name":"Dos",

    "pjone":"234567891"

}

{

    "email":"tres@gmail.com",

    "imageUrl":"https://foto.com/tres",

    "jobTitle":"UI/UX",

    "name":"Tres",

    "pjone":"345678912"

}

{

    "email":"cuatro@gmail.com",

    "imageUrl":"https://foto.com/cuatro",

    "jobTitle":"UI/UX",

    "name":"Cuatro",

    "pjone":"456789123"

}

# Best Practices in Angular

Antes de arrancar con la segunda parte del tutorial de amigoscode, que se ocupa del frontend, pienso que conviene estudiar este artículo:

<https://itnext.io/best-practices-in-angular-a8926fa02ae2>

Muy claro, simple, directo al grano. Pienso que es un buen lugar para comenzar.

En particular, prestar atención a la explicación del principio LIFT.

## LIFT Principle

Let’s take a look at the LIFT principle:

**L**ocating our code is easy.

**I**dentify code at a glance.

**F**lat structure as long as we can.

**T**ry to stay DRY.

This principle helps you to find code quickly. So, if you don’t use this principle, ask yourself: How quickly can you open and work in all of the files that you need in order to build a feature? My advice: Respect and use LIFT.

## Building an Angular Application Structure

<https://medium.com/god-dev-blog/building-an-angular-application-structure-ef4cee9da934>

El artículo que más me gustó, entre muchos que revisé. Lo que quiero que hagan ahora es pegarle una ojeada, de modo que les queden en la memoria los términos que usa y los temas de los que habla. Cuando tengan una aplicación que empieza a crecer y ser difícil de manejar, será el momento de estudiar este artículo a fondo.

# Frontend en Angular

Ahora sí, seguimos con el tutorial de amigoscode.

import org.springframework.web.bind.annotation.CrossOrigin;

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