FRONTEND BACKEND PICAPIEDRA

# Backend

API REST con SpringBoot. El backend es el servidor, que corre en un server, potencialmente al otro lado del mundo. Sirve los microservicios que va a consumir el frontend, que corre en la computadora del cliente.

# Frontend

Angular. Este es el frontend, que corre en la computadora del cliente. Consume los microservicios que sirve el server backend, que corre potencialmente en un server al otro lado del mundo.

# Aplicación

El frontend consume los servicios del backend.

Envía los requests usando Http con el módulo de HTTPCLIENT de Angular.

El backend atiende los requests que el frontend manda por HPPT.

# Proyecto para el backend

Generar un proyecto Spring Boot.

Java: Create Java Project…

Spring Boot

Maven Project

2.6.7

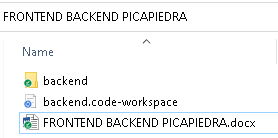
Java

tup

backend

jar

11

Spring Web

Guardar en la carpeta en la que está este documento.

Open

File > Save Workspace As

Navegar hasta la carpeta que contiene el proyecto backend, no dentro de la carpeta backend, sino la carpeta que la contiene.

# 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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.6.7</version>

        <relativePath/> <!-- lookup parent from repository -->

    </parent>

    <groupId>tup</groupId>

    <artifactId>backend</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>backend</name>

    <description>Demo project for Spring Boot</description>

    <properties>

        <java.version>11</java.version>

    </properties>

    <dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

        </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>

Como se puede apreciar, el proyecto es lo más simple posible. La dependencia web es para poder usar el browser.

# src\main\resources\application.properties

Queda vacío, no se modifica para nada.

# src\main\java\tup\backend\BackendApplication.java

package tup.backend;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BackendApplication {

    public static void main(String[] args) {

        SpringApplication.run(BackendApplication.class, args);

    }

}

Esta es la clase principal, tal como se genera automáticamente. No necesitamos modificarla.

# src\main\java\tup\backend\configuration\WebConfiguration.java

package tup.backend.configuration;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebConfiguration implements WebMvcConfigurer {

    @Override

    public void addCorsMappings(CorsRegistry registry) {

        registry.addMapping("/\*\*").allowedMethods("\*");

    }

}

Necesitamos esta clase por las restricciones CORS. El backend percibe los requests del frontend como proviniendo de otro dominio.

La solución que usé está tomada de:

<https://stackoverflow.com/questions/44697883/can-you-completely-disable-cors-support-in-spring>

Este artículo es milagrosamente claro y explícito. El anterior simplemente da la solución. Este explica todo:

<https://howtodoinjava.com/spring-boot2/spring-cors-configuration/>

La documentación oficial está en:

<https://spring.io/blog/2015/06/08/cors-support-in-spring-framework#javaconfig>

Este artículo de la Wikipedia discute CORS en general:

<https://en.wikipedia.org/wiki/Cross-origin_resource_sharing>

La traducción de ese artículo al español es horrenda. Mi conciencia no me permite poner esa chanchada en el material de estudio.

# src\main\java\tup\backend\controller\Controller.java

package tup.backend.controller;

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

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

@RestController

public class Controller {

    @GetMapping()

    public String home() {

        return "Home";

    }

    @GetMapping("/nombre")

    public String nombre() {

        return "Esta parte de la aplicación se llama backend.";

    }

}

Es es la única clase controladora de nuestro backend picapiedra. Tiene solo dos métodos GET.

Y eso es todo. Hay que generar la aplicación, y hacerla correr. Recordar que va a estar en el puerto 8080. Si el backend no está corriendo, el frontend no recibe respuesta.

# Proyecto para el frontend

Se necesita tener instalado Angular en la PC, además de las extensiones de Angular para el VS Code.

Yo instalé Angular Extension Packv1.0.0, de Loiane Groner, que tiene un montón de extensiones incluidas.

Abrir una consola de Windows en la carpeta que contiene al proyecto backend. No dentro de la carpeta backend, sino en la carpeta que contiene a backend.

ng new frontend

Pregunta si queremos Angular routing: Yes

Hojas de estilo: elegir SCSS usando los cursores.

Esperar a que termine.

Ir a VS Code, donde tenemos abierto el proyecto del backend. En el menú File, Agregar carpeta al workspace. Agregar frontend. Grabar el workspace en la carpeta que contiene a ambos proyectos.

El proyecto de Angular aparece en la pestaña del workspace. En la de proyectos de Java aparece, pero no como un proyecto, sino simplemente como carpeta. Hay que usar la pestaña superior.

# src\index.html

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Frontend</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>

El index.html, tal como fue generado. Notar la forma de poner el favicon, y el componente principal de la aplicación.

# src\app\app.component.html

<h1>{{ title }}</h1>

<p>{{ unParrafo }}</p>

<p>{{ respuesta }}</p>

<router-outlet></router-outlet>

Borrar todo el contenido del archivo, y pegar lo anterior.

La sintaxis {{}} significa que lo que está entre las dobles llaves es una variable, y en el momento de renderizar su valor se pone en ese placeholder. En este caso, tenemos tres variables: title, unParrafo y respuesta.

RouterOutlet[[1]](#footnote-1) es una directiva de Angular. Actúa como un marcador de posición que Angular llena dinámicamente según el estado actual del enrutador.

# src\app\app.component.ts

Pegar el siguiente código:

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

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.scss'],

})

export class AppComponent {

  title = 'Header 1, título de la página';

  unParrafo = 'Este es un párrafo de muestra.';

  respuesta = '';

  constructor(private http: HttpClient) {}

  ngOnInit() {

    this.http

      .get('http://localhost:8080/nombre', { responseType: 'text' })

      .subscribe((resp: any) => {

        this.respuesta = resp;

      }),

      (error: any) => {

        console.log(error);

      };

  }

}

Este es un archivo con extensión ts, que significa TypeScript, que es un superset de JavaScript.

Se puede ver cómo define las tres variables que luego se renderizan en app.component.html.

El constructor lo tuvimos que agregar, porque es obligatorio.

También agregamos el método ngOnInit() que, como su nombre lo indica, se ejecuta al inicializar. Dentro de él ponemos nuestros requests.

# src\app\app.module.ts

Pegar el siguiente código:

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

import { BrowserModule } from '@angular/platform-browser';

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

@NgModule({

  declarations: [AppComponent],

  imports: [

    BrowserModule,

    AppRoutingModule,

    // import HttpClientModule after BrowserModule.

    HttpClientModule,

  ],

  providers: [],

  bootstrap: [AppComponent],

})

export class AppModule {}

Las líneas que hemos agregado son:

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

y

    // import HttpClientModule after BrowserModule.

    HttpClientModule,

que nos permiten hacer los requests por HTTP.

# Arrancar el frontend

Dentro de VS Code hacemos click derecho sobre la carpeta del frontend, y seleccionamos abrir en la terminal built-in. En la ventana de la terminal escribimos:

ng serve

Si aparece el mensaje:

Your global Angular CLI version (13.3.5) is greater than your local version (13.3.0). The local Angular CLI version is used.

Ejecutar el comando:

npm i --save--dev @angular/cli@latest

Y listo.

El backend y el frontend los veremos corriendo respectivamente en:

<http://localhost:8080>

y

<http://localhost:4200>

# Execution Policy

En PowerShell elevado

get-executionpolicy (probablemente da restricted)

set-executionpolicy unrestricted

1. <https://angular.io/api/router/RouterOutlet> [↑](#footnote-ref-1)