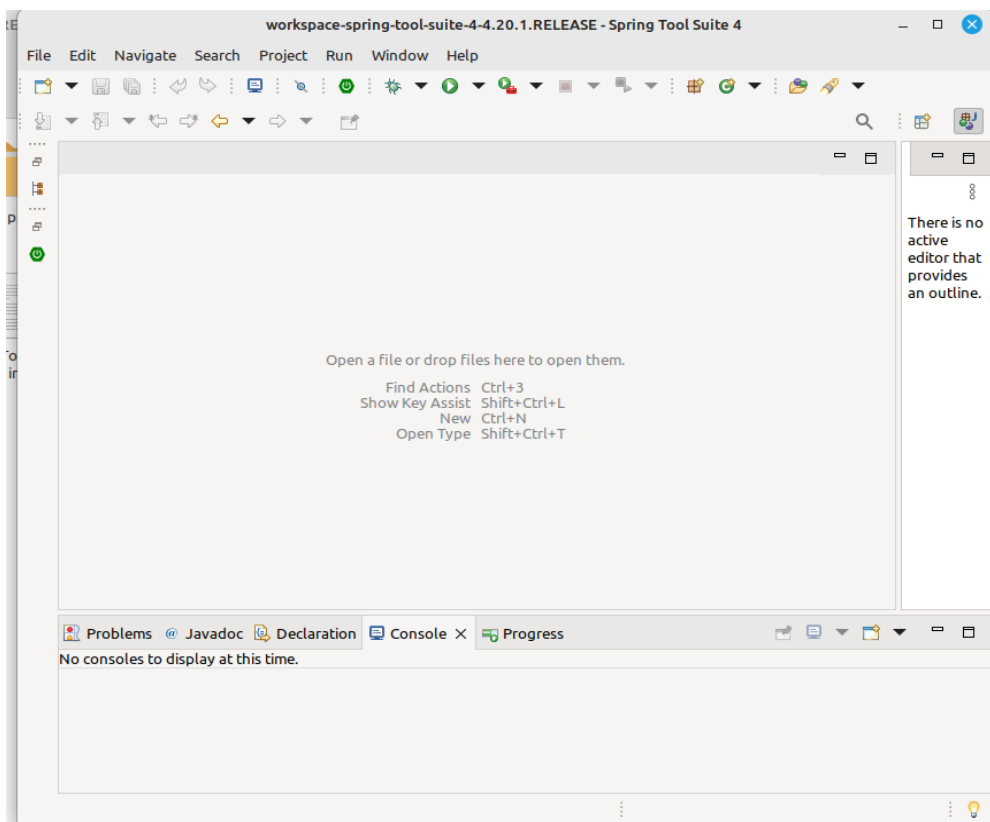
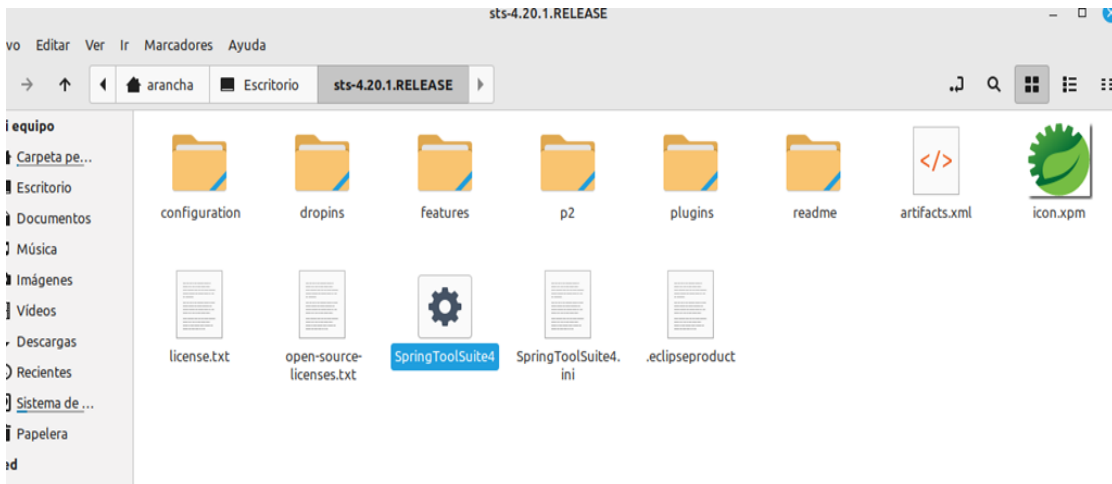


## Tarea 1 – UD4

### Toma de contacto con SpringBoot

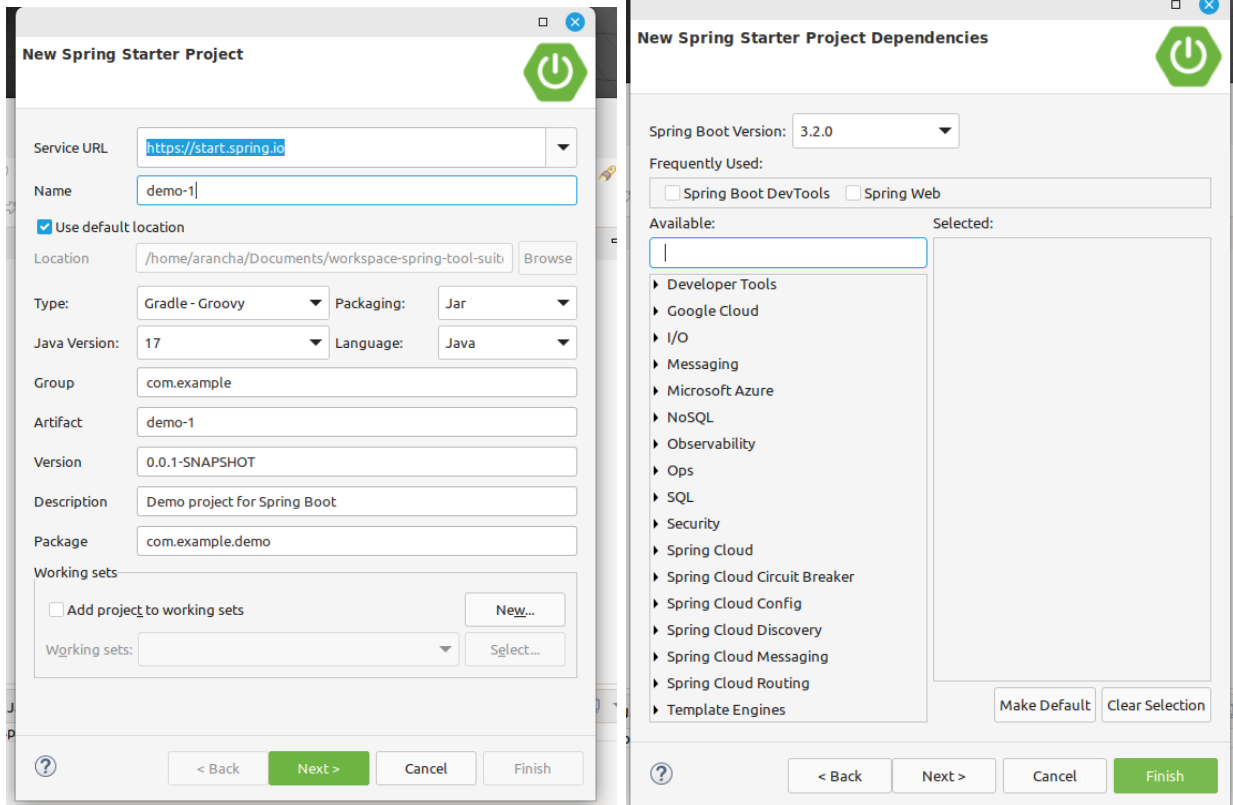
#### Descripción de la Práctica:

##### 1. DESCARGA STS 4 Eclipse y ejecutar:

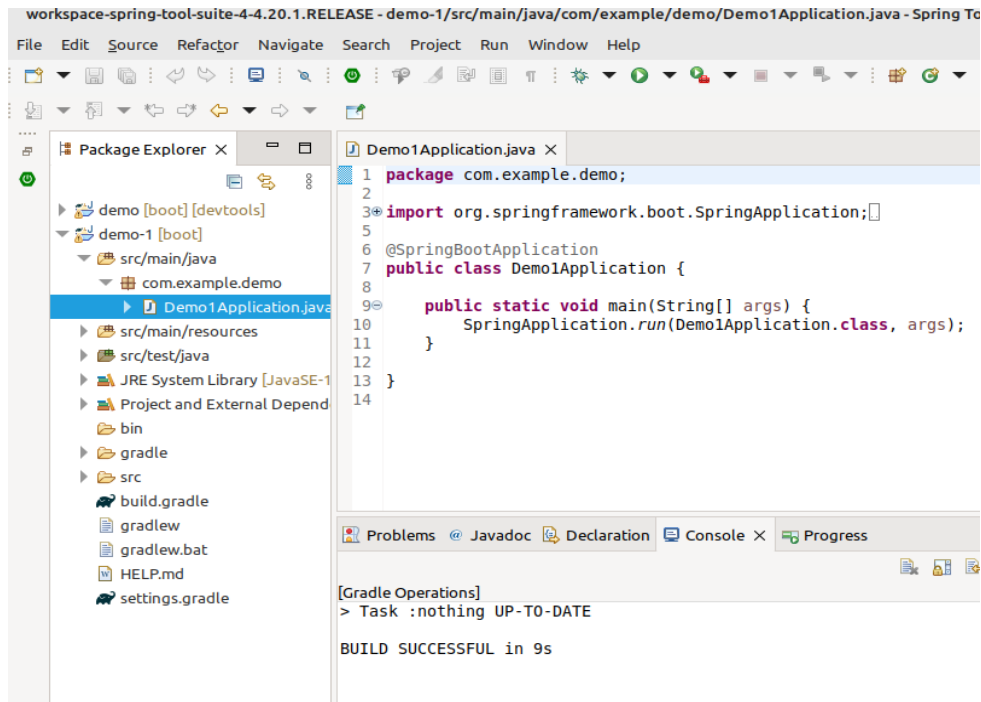


## 2. PRIMER INTENTO DE NUEVO PROYECTO:

- En STS → Create new Spring Starter Project.
- Dejamos todos los valores por defecto. (me fijo sobretodo en el packagin y language, Jar y Java, pues en lenguaje java es el que vamos a usar y jar es tipo de ejecutable para java).
- Next sin seleccionar dependencia.

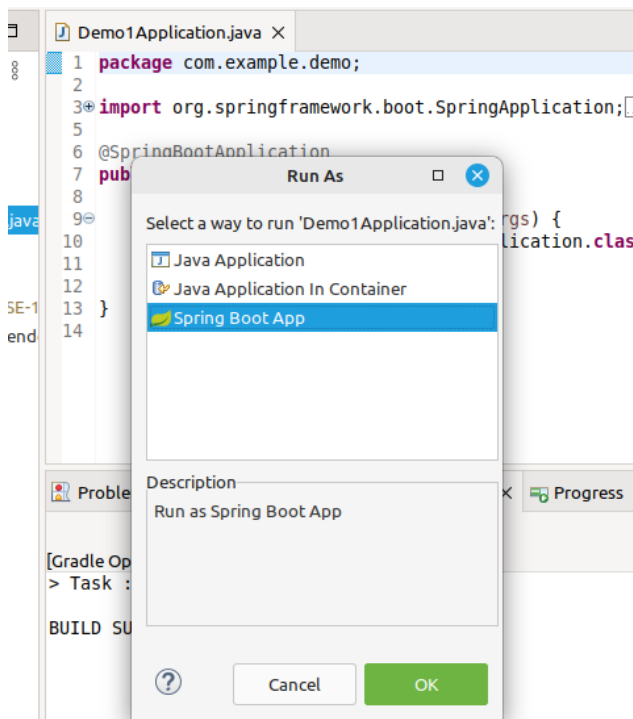


- Vemos el código fuente creado, solo DemoApplication.java en `src/main/java/com/example/demo/`



- Si damos a Run nos pide varias opciones, la última Spring Boot App, le damos. Lanza pero no hace nada ¿Se puede parar o terminó?

**Se lanza pero termina en el mismo instante.**



```

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class Demo1Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Demo1Application.class, args);
11     }
12 }
13
14

```

Console Output:

```

<terminated> demo-1 - Demo1Application [Spring Boot App] /home/arancha/Escritorio/sts-4.20.1.RELEASE/plugins/org.eclipse.jdt.launcher
:: Spring Boot :: (v3.2.0)
2023-12-12T10:39:05.735+01:00 INFO 8264 --- [main] com.example.demo.Demo1Application
2023-12-12T10:39:05.739+01:00 INFO 8264 --- [main] com.example.demo.Demo1Application
2023-12-12T10:39:06.108+01:00 INFO 8264 --- [main] com.example.demo.Demo1Application

```

- Compara el código con el de DemoApplication.java aquí: <https://spring.io/quickstart> y añade las líneas que falten ¿funciona? ¿localiza bien los imports de las anotaciones?  
**No funciona, salen errores en las anotaciones y no localiza los imports. Al posar el ratón encima, no reconoce los imports.**

Conclusión: intento fallido.

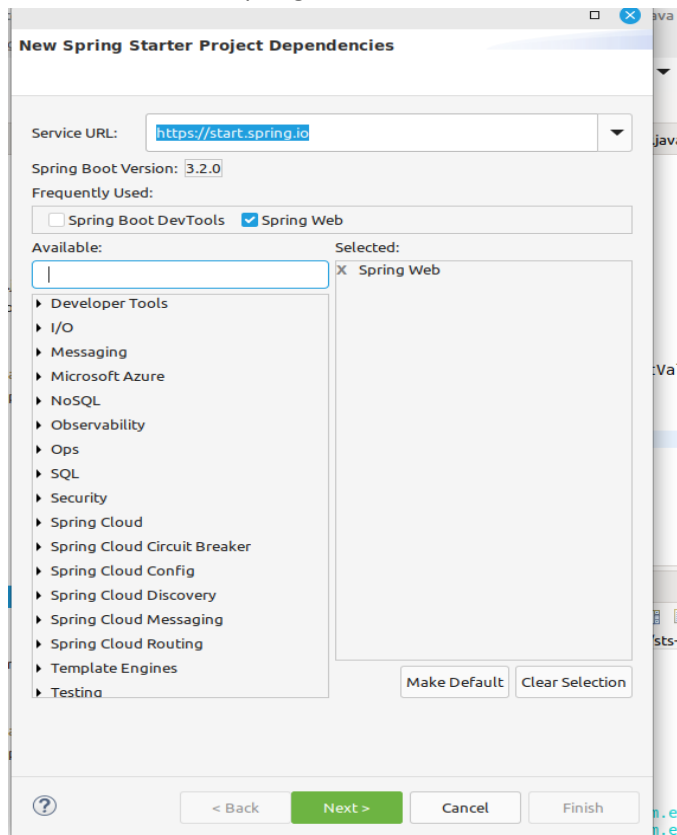
```

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class Demo1Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Demo1Application.class, args);
11     }
12
13     @GetMapping("/hello")
14     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
15         return String.format("Hello %s!", name);
16     }
17 }
18

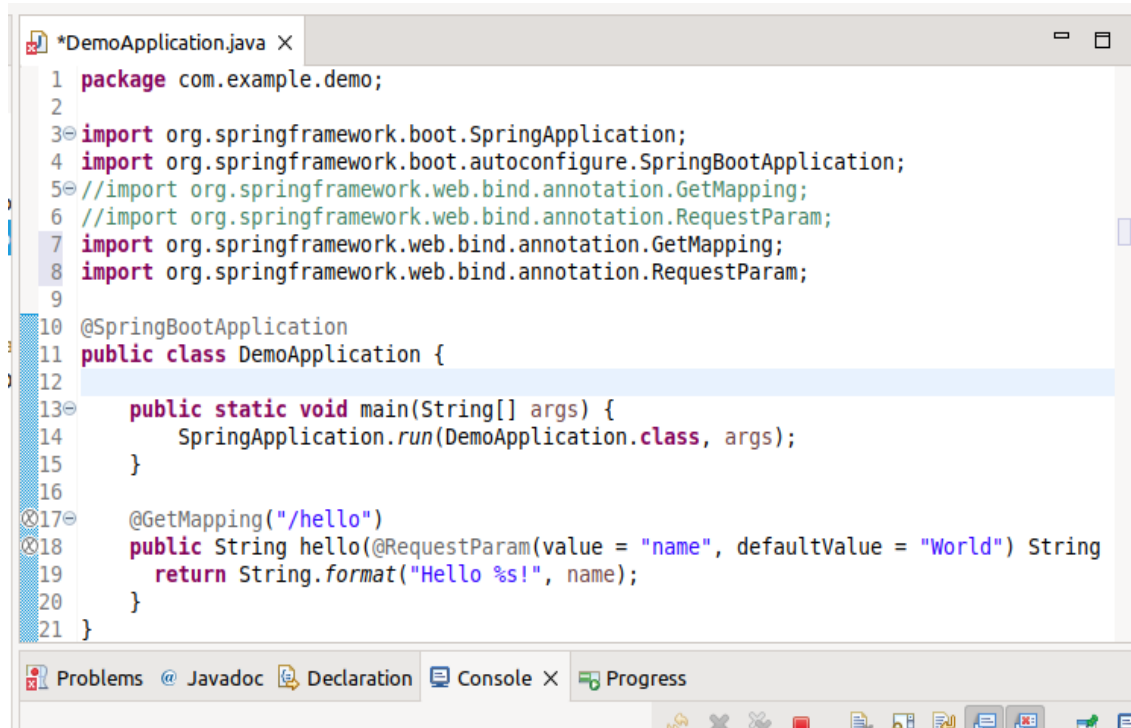
```

### 3. SEGUNDO INTENTO DE NUEVO PROYECTO:

- Borra el proyecto anterior y vuelve a crearlo con los mismos pasos PERO en dependencias selecciona Web → Spring Web:



- ¿Funcionan ahora las anotaciones y los imports? ¿Si quitas los imports propone esos mismos u otros? **Ahora añade los imports automáticamente y al probar a borrarlos (comentarlos), vemos que al posarnos encima de las dependencias, proponen los mismos:**



- ¿Qué pasa ahora al dar al botón verde de iniciar? **Vemos que se ejecuta correctamente y queda en ejecución, y nos da la opción de poder pararlo.**

The screenshot shows an IDE with the following code in `Demo1Application.java`:

```

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7
8 @SpringBootApplication
9 public class Demo1Application {
10
11     public static void main(String[] args) {
12         SpringApplication.run(Demo1Application.class, args);
13     }
14
15     @GetMapping("/hello")
16     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
17         return String.format("Hello %s!", name);
18     }
19 }
20

```

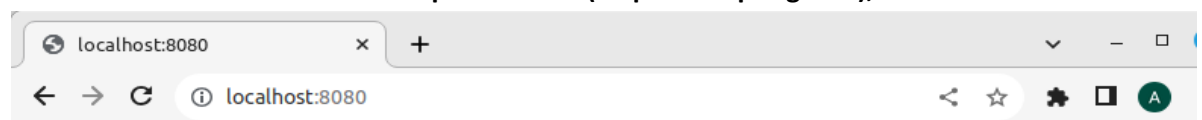
The console output shows the application starting successfully:

```

demo-1 - Demo1Application [Spring Boot App] [pid: 18475]
:: Spring Boot :: (v3.2.0)
2023-12-12T10:56:54.251+01:00 INFO 18475 --- [main] com.example.demo.Demo1Application
2023-12-12T10:56:54.261+01:00 INFO 18475 --- [main] com.example.demo.Demo1Application
2023-12-12T10:56:54.907+01:00 INFO 18475 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2023-12-12T10:56:54.913+01:00 INFO 18475 --- [main] o.apache.catalina.core.StandardServer
2023-12-12T10:56:54.913+01:00 INFO 18475 --- [main] o.apache.catalina.core.StandardEngine
2023-12-12T10:56:54.943+01:00 INFO 18475 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/]
2023-12-12T10:56:54.943+01:00 INFO 18475 --- [main] w.s.c.ServletWebServerApplicationContext
2023-12-12T10:56:55.245+01:00 INFO 18475 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2023-12-12T10:56:55.250+01:00 INFO 18475 --- [main] com.example.demo.Demo1Application

```

- Accede con el navegador web a localhost al puerto indicado ¿qué pasa?  
**Accediendo desde localhost con el puerto 8080 (el que usa springboot), sale error:**



## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Dec 12 11:03:59 CET 2023

There was an unexpected error (type=Not Found, status=404).

No static resource .

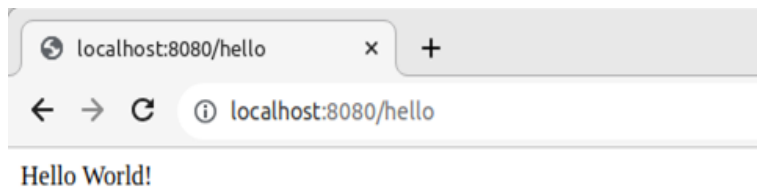
org.springframework.web.servlet.resource.NoResourceFoundException: No static resource .

at

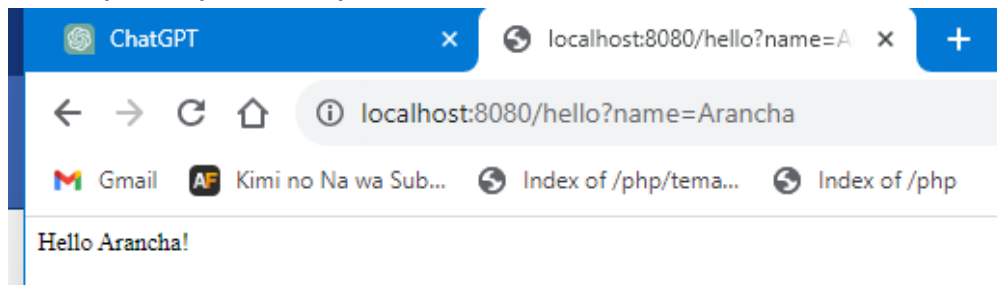
org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:111)

Que es porque no le estoy pasando la url que se mapea/conecta en el método que queremos probar.

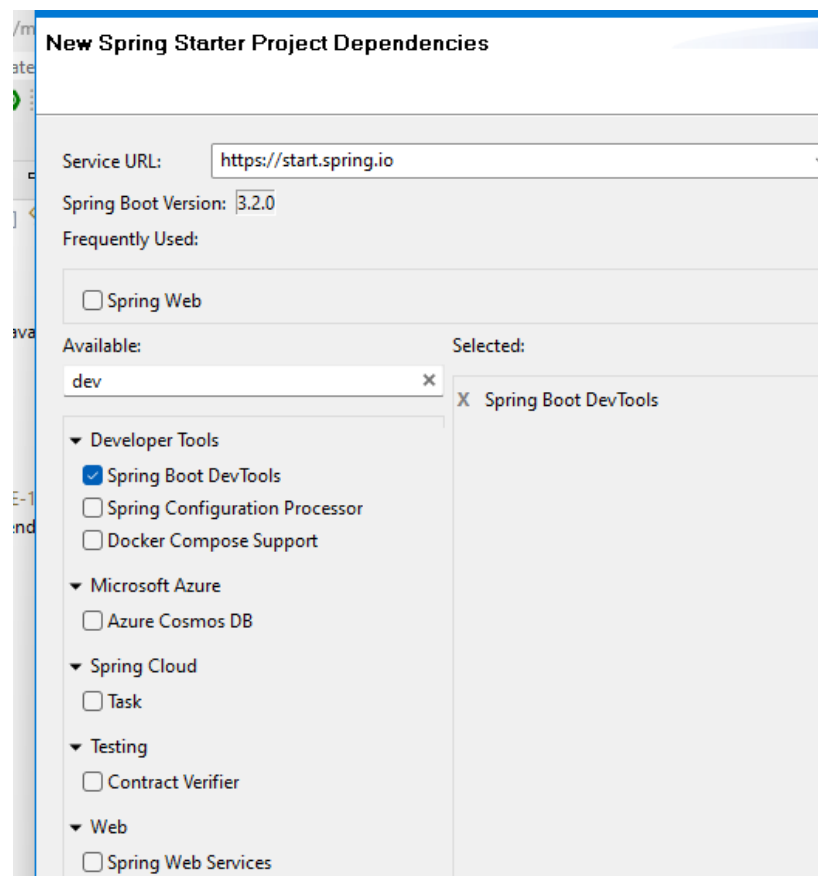
- Prueba poniendo la URL `http://localhost:8080/hello` y también pasándole un parámetro name.  
**Pongo hello en la url y sale en el navegador el mensaje correspondiente en el return del método hello de nuestra clase:**  
**\*\*\*Importante: añadir la anotación @RestController encima de la clase.**



Y ahora pruebo pasando el parámetro name con valor Arancha:

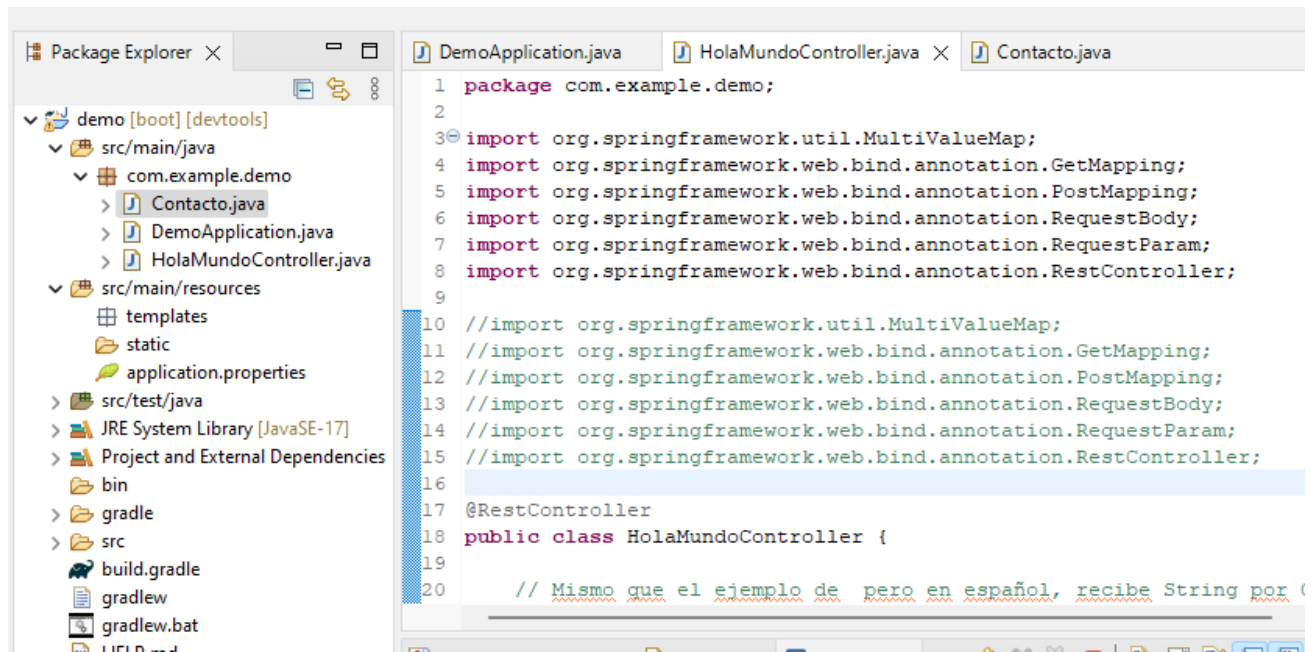


- Añade la dependencia Spring Boot DevTools para que reinicie el servidor cada vez que cambias y guardas código. Recuerda cómo lo hizo el profesor en clase: botón derecho → Spring → Add Starters ...



### 3. PRUEBAS. Copia las clases HolaMundoController y Contacto:

- ¿Qué pasa con las anotaciones y los imports? ¿Si quitas los imports propone esos mismos u otros?  
**Copio las clases indicadas y no sale ningún error. Como ya habíamos añadido al proyecto la dependencia SpringWeb, los imports se añaden automáticamente. Pruebo a comentarlos y efectivamente, al posarnos sobre las anotaciones proponen los mismos imports.**

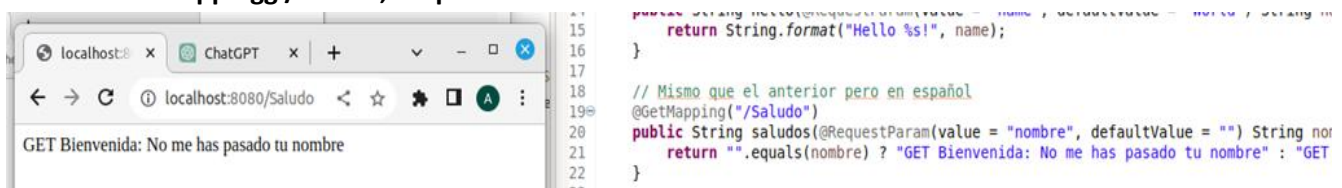


- Haz pruebas desde el navegador contra los puntos de acceso creados en el código de HolaMundoController (GetMapping): **voy haciendo pruebas en el navegador de todos los métodos de tipo GET.**

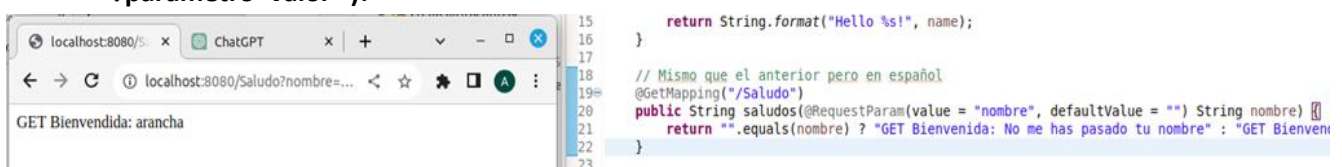
En cada prueba, a la izquierda se ve la url y salida del navegador, y a la derecha la parte del código correspondiente, el método que uso para la prueba.

**\*\* Pruebas hechas con los docs iniciales, antes de las modificaciones del profesor:**

- Getmappingg /Saludo, sin parámetro:



- Getmapping /Saludo, con parámetro nombre (al poner el parámetro, se pone "?parámetro=valor"):



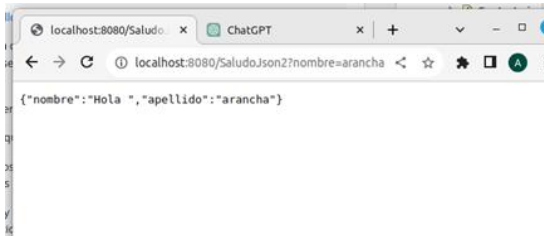


**- Getmapping / SaludoJson2 sin parámetro:**

```

34         : "BienvenidaJSON: " + persona.getNombre() + " " + persona.getApellido();
35     }
36
37     // Por GET recibiendo String y devolviendo JSON
38     @GetMapping("/SaludoJson2")
39     public Contacto envioJson2(@RequestParam(value = "nombre", defaultValue = "Mundo") String nombre) {
40         return new Contacto("Hola ", nombre);
41     }
42

```

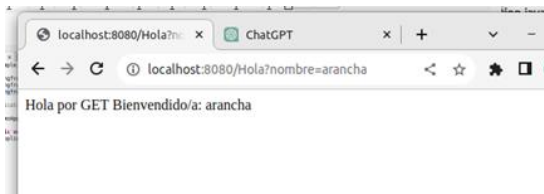
**- Getmapping / SaludoJson2 con parámetro:**

```

10 // Por POST recibiendo Json y devolviendo String
11 @PostMapping("/SaludoJson")
12 public String envioJson(@RequestBody Contacto persona) {
13     return " " + persona.getNombre() + " " + persona.getApellido();
14 }
15
16 // Por GET recibiendo String y devolviendo JSON
17 @GetMapping("/SaludoJson2")
18 public Contacto envioJson2(@RequestParam(value = "nombre", defaultValue = "Mundo") String nombre) {
19     return new Contacto("Hola ", nombre);
20 }
21
22 // Por POST recibiendo JSON y devolviendo JSON
23

```

*Ahora pruebas con el archivo nuevo del profesor:*

**-Getmapping /Hola con parámetro nombre:**

```

8 import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 public class HelloWorldController {
12
13     // Mismo que el ejemplo de pero en español, recibe String por GET y devuelve String
14     @GetMapping("/Hola")
15     public String holaGet1(@RequestParam(value = "nombre", defaultValue = "") String nombre) {
16         if (nombre==null || "".equals(nombre))
17             return "Hola por GET: No me has pasado tu nombre";
18         else
19             return "Hola por GET Bienvenido/a: " + nombre;
20     }
21

```

**-Getmapping /HolaJson sin parámetro:**

```

16         if (nombre==null || "".equals(nombre))
17             return "Hola por GET: No me has pasado tu nombre";
18         else
19             return "Hola por GET Bienvenido/a: " + nombre;
20     }
21
22     // No tiene sentido recibir JSON por GET, complica todo mucho y no se usa
23
24     // Por GET recibiendo String y devolviendo JSON
25     @GetMapping("/HolaJson")
26     public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre") String nombre,
27                             @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apellido) {
28         return new Contacto(nombre, apellido);
29     }
30

```

**-Getmapping /HolaJson con un parámetro nombre:**

```

16         if (nombre==null || "".equals(nombre))
17             return "Hola por GET: No me has pasado tu nombre";
18         else
19             return "Hola por GET Bienvenido/a: " + nombre;
20     }
21
22     // No tiene sentido recibir JSON por GET, complica todo mucho y no se usa
23
24     // Por GET recibiendo String y devolviendo JSON
25     @GetMapping("/HolaJson")
26     public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre") String nombre,
27                             @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apellido) {
28         return new Contacto(nombre, apellido);
29     }
30

```

**-Getmapping /HolaJson con dos parámetros nombre y apellido (en este caso, para poner varios parámetros hay que poner "&" entre cada uno):**

```

30         if (nombre==null || "".equals(nombre))
31             return "Hola por GET: No me has pasado tu nombre";
32         else
33             return "Hola por GET Bienvenido/a: " + nombre;
34     }
35
36     // No tiene sentido recibir JSON por GET, complica todo mucho y no se usa
37
38     // Por GET recibiendo String y devolviendo JSON
39     @GetMapping("/HolaJson")
40     public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre") String nombre,
41                             @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apellido) {
42         return new Contacto(nombre, apellido);
43     }
44

```

Ahora para los métodos de POST he usado Postman, herramienta bastante intuitiva y fácil de usar:

Como vamos a probar métodos POST, seleccionamos POST, y en el campo de la ruta, la url de nuestro PostMapping correspondiente.

### -Postmapping /Hola:

En el apartado Body, seleccionamos opción x-www-form-urlencoded y ahora en Key ponemos el nombre del parámetro que queremos usar, y en value, el valor de dicho parámetro (aquí podríamos añadir tantos parámetros/valores como tenga nuestro método). Y vemos en el campo Body inferior, la salida por pantalla (el equivalente a la salida por el navegador en los métodos GET)

The screenshot shows a Postman interface for a POST request to `localhost:8080/Hola` with the body type set to `x-www-form-urlencoded`. The key-value pair is `nombre: arancha`. The response body shows `1 Hola por POST. Bienvenido/a: arancha`.

The IDE shows the following Java code:

```

@PostMapping("/Hola")
public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre")
    @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apel
    ) {
    return new Contacto(nombre,apellido);
}

// Ahora por POST, recibiendo un String con el nombre y devolviendo un String
// RequestBody en este caso recibe todos los parámetros pasados en el cuerpo de la p
// POST con x-www-form-urlencoded
@PostMapping("/Hola")
public String holaPost1(@RequestBody(required=false) MultiValueMap<String,String> par
    if (paramMap==null)
        return "Hola por POST. No me has pasado tu nombre";
    else
        return "Hola por POST. Bienvenido/a: " + paramMap.getFirst("nombre");
}

// Por POST recibiendo Json y devolviendo String
@PostMapping("/HolaJson")

```

The console output shows:

```

023-12-12T12:37:38.024+01:00 INFO 47630 --- [ restartedMain] .ConditionEvaluationDeltaLogg
023-12-12T12:37:41.125+01:00 INFO 47630 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localh
023-12-12T12:37:41.126+01:00 INFO 47630 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherSer
023-12-12T12:37:41.126+01:00 INFO 47630 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherSer
023-12-12T12:44:14.829+01:00 WARN 47630 --- [io-8080-exec-10] .w.s.m.s.DefaultHandlerExcept

```

### - Postmapping /HolaJson:

En este caso, al tener que introducir un Json, tenemos que seleccionar raw y luego JSON, y en el campo de texto introducimos en formato Json, nuestros parámetros con sus valores tal y como se muestran en la imagen. Y en campo inferior vemos la salida por consola.

The screenshot shows a Postman interface for a POST request to `localhost:8080/HolaJson` with the body type set to `raw` and `JSON`. The body content is `{ "nombre": "arancha", "apellido": "chicharro" }`. The response body shows `1 Hola POST JSON. Bienvenido/a: arancha chicharro`.

The IDE shows the following Java code:

```

@PostMapping("/Hola")
public String holaPost1(@RequestBody(required=false) MultiValueMap<String,String>
    if (paramMap==null)
        return "Hola por POST. No me has pasado tu nombre";
    else
        return "Hola por POST. Bienvenido/a: " + paramMap.getFirst("nombre");
}

// Por POST recibiendo Json y devolviendo String
@PostMapping("/HolaJson")
public String holaPost2(@RequestBody(required=false) Contacto persona) {
    if (persona == null)
        return "Hola POST JSON. No has pasado contacto";
    else
        return "Hola POST JSON. Bienvenido/a: " + persona.getNombre() + " " + per
}

// Por POST recibiendo JSON y devolviendo JSON
@PostMapping("/HolaJson2")
public Contacto holaPost3(@RequestBody(required=false) Contacto persona) {

```

The console output shows:

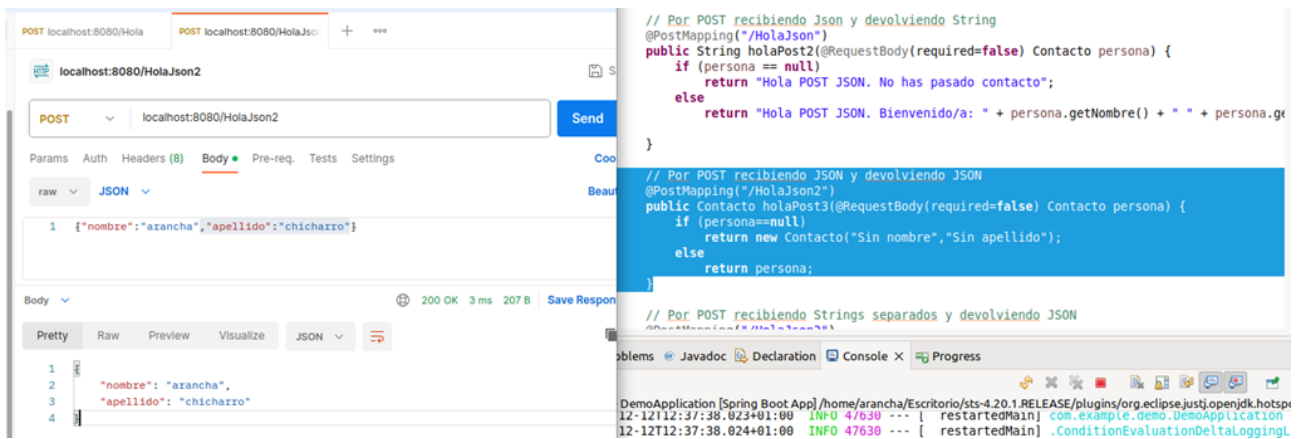
```

023-12-12T12:37:38.024+01:00 INFO 47630 --- [ restartedMain] .ConditionEvaluationDeltaLogg
023-12-12T12:37:41.125+01:00 INFO 47630 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localh
023-12-12T12:37:41.126+01:00 INFO 47630 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherSer
023-12-12T12:37:41.126+01:00 INFO 47630 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherSer
023-12-12T12:44:14.829+01:00 WARN 47630 --- [io-8080-exec-10] .w.s.m.s.DefaultHandlerExcept

```

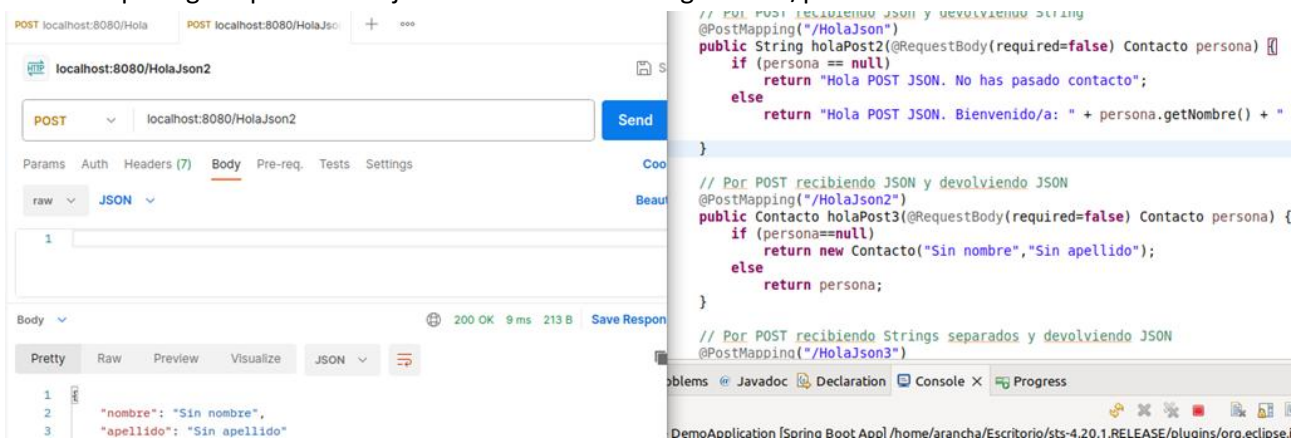
### -Postmapping /HolaJson2:

Este caso es igual que el anterior, con la diferencia que en consola, el campo Body inferior, devuelve un Json en lugar de un texto.



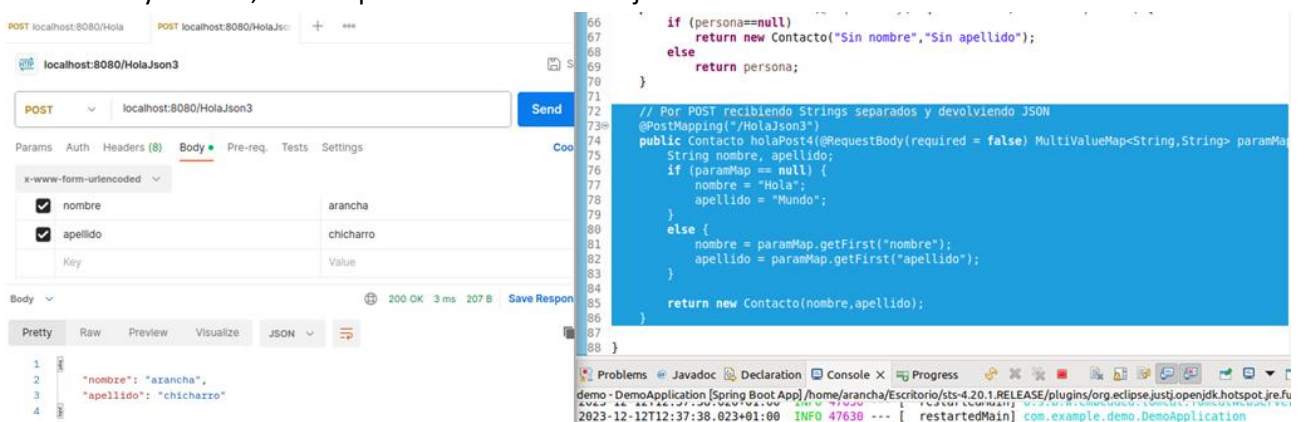
### - Postmapping /HolaJson2 sin parámetros:

Aquí hago la prueba de ejecutar url sin enviar ningún valor/parámetro:



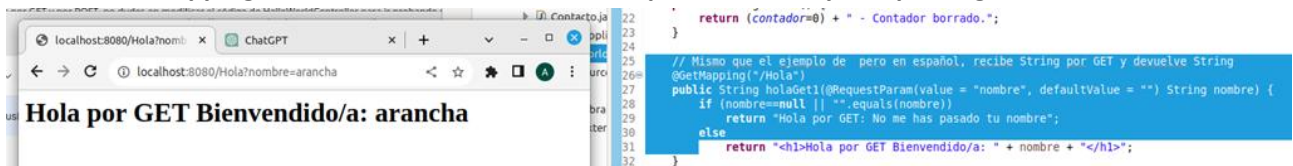
### -Postmapping /HolaJson3:

En esta prueba, introducimos valores por Strings y nos devuelve Json. Lo hacemos como en la primera prueba, seleccionando x-www-form-urlencoded y en los campos key ponemos los dos parámetros que vamos a enviar, y en los campos value sus valores. Y por lo tanto en el campo de Body inferior, vemos que nos devuelve un objeto Json.

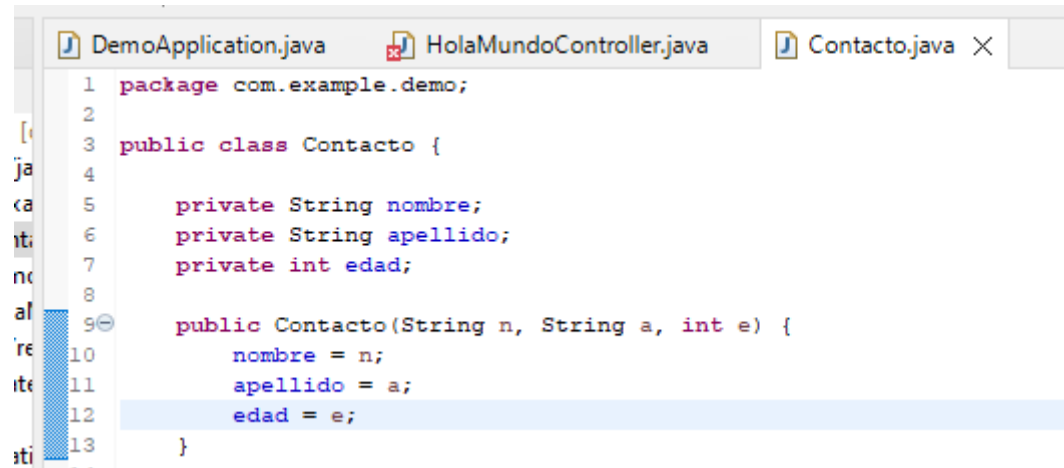


**\*\*Pruebas extra:**

-GetMapping /Hola: añadido en el return la etiqueta html h1, para que salga en formato título:



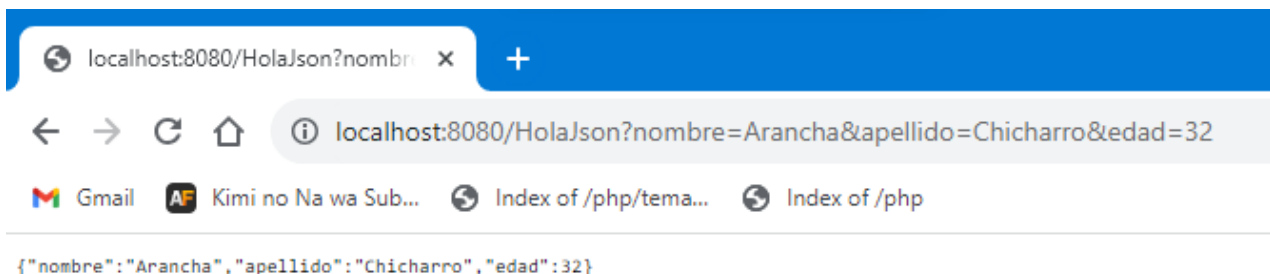
-GetMapping /HolaJson: añadido un tercer parámetro edad. Para ello en la clase Contacto añadido nuevo atributo int edad, añadiéndolo también en el constructor y su get y set:



Y en la clase HolaMundoController, en el método HolaJson añadimos un nuevo `@RequestParam` para el parámetro edad, añadiendo también en el return. Quedaría así:

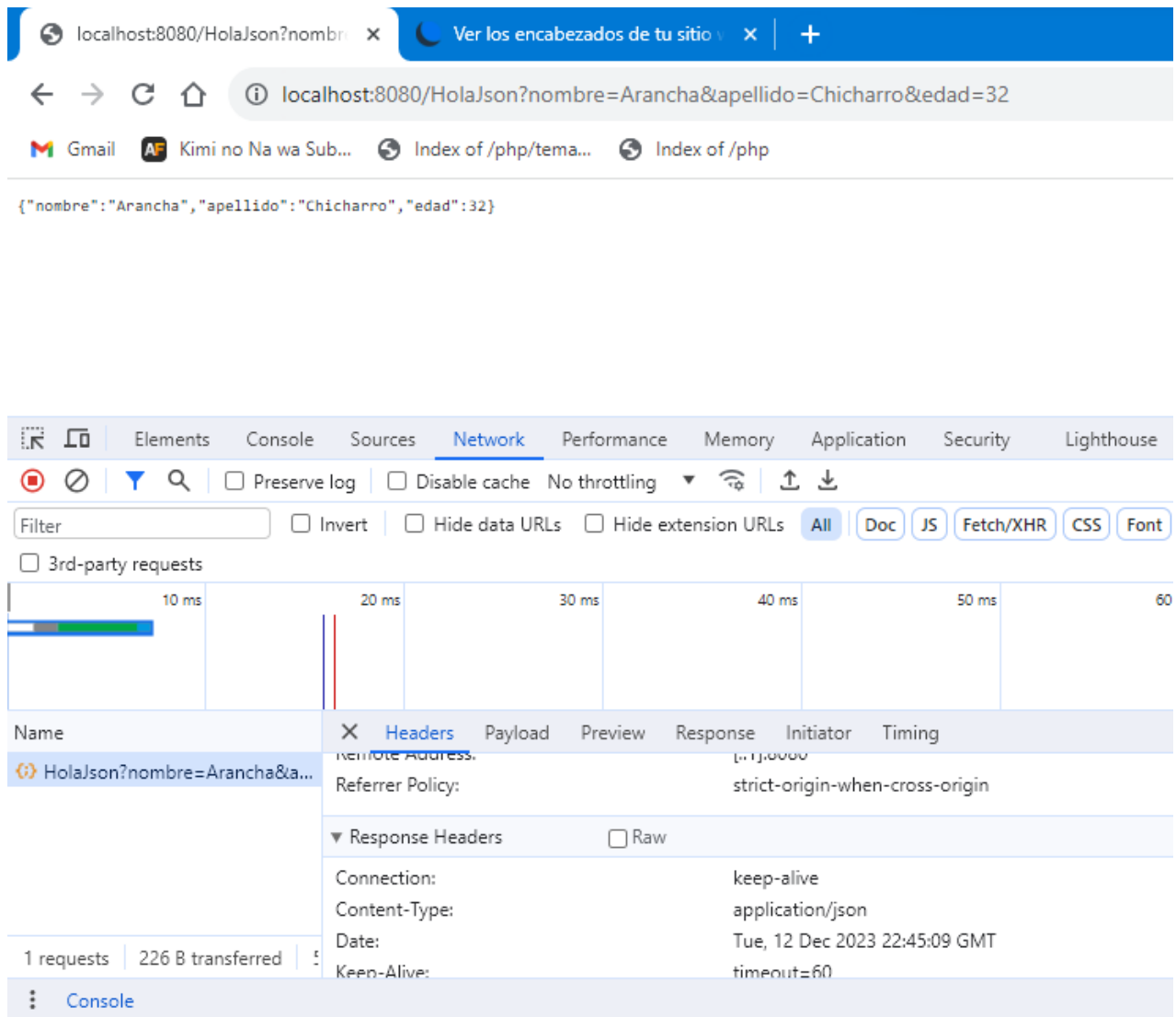
```
// Por GET recibiendo String y devolviendo JSON
@GetMapping("/HolaJson")
public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre") String nombre,
    @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apellido,
    @RequestParam(value = "edad", defaultValue = "Sin edad") int edad) {
    return new Contacto(nombre, apellido, edad);
}
```

Prueba del navegador, para poner los tres parámetros, hay que separarlos con el carácter "&":



Aprovecho para, desde el navegador, ver las cabeceras y buscar `ContentType`, vemos que es json:



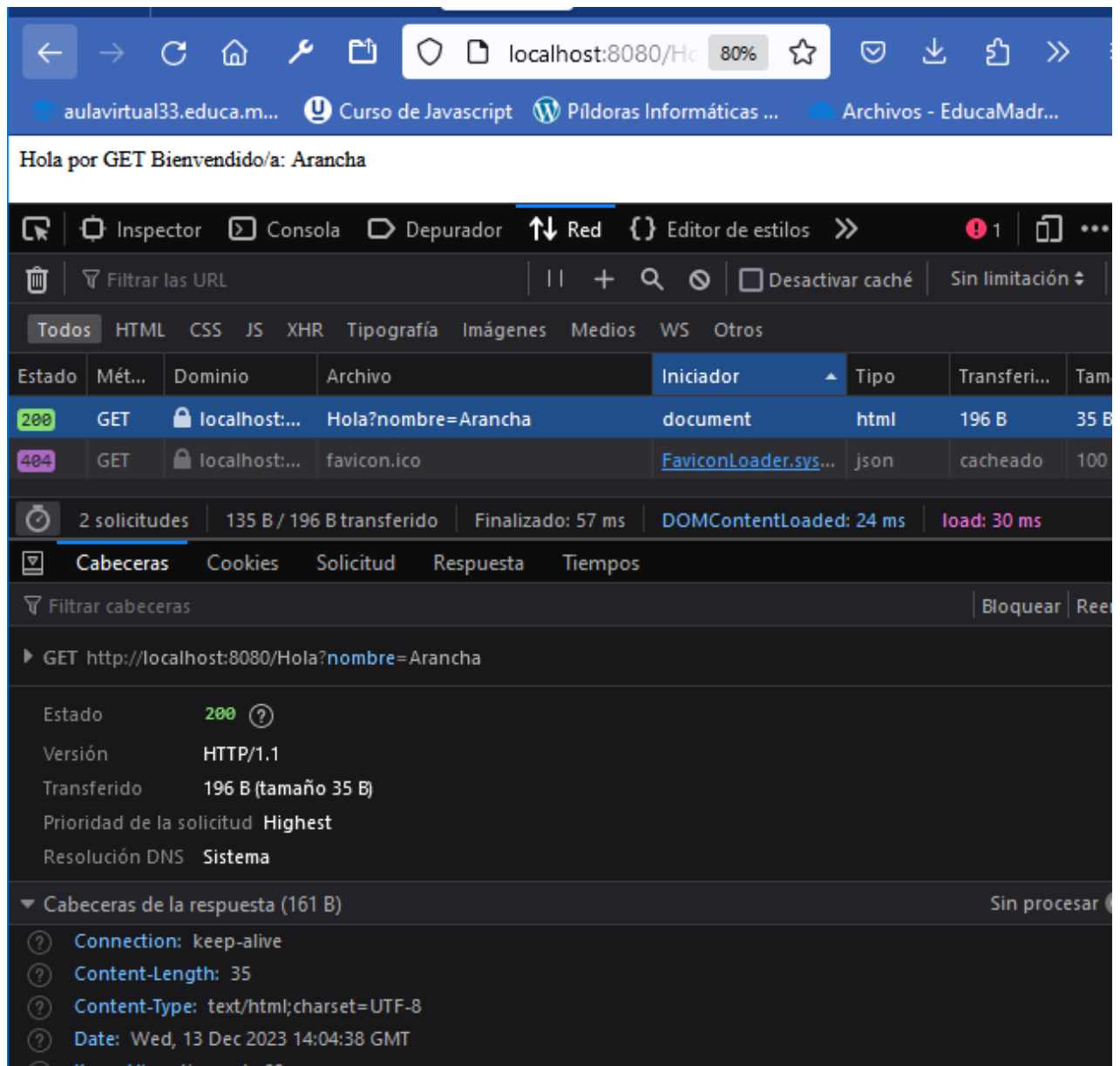


- Prueba a responder con HTML ¿lo ve bien el navegador? Localiza el Content-type de la respuesta.  
**Avanzado:** ¿Dónde se gestiona el Content-type de la respuesta en el código Java?  
*En esta pregunta, entiendo que debo revisar la respuesta de la petición que he mandado al navegador y verificar el contenido además de localizar el Content-type.*

Hago la prueba con el método `GetMapping /Hola` con parámetro.

En inspeccionar, vemos en el apartado Red, seleccionamos la petición que hemos enviado, GET claro, y vemos que es de Tipo html, y en las Cabeceras, en cabeceras de la respuestas podemos localizar el Content-Type y vemos que es `text/html`, es correcto pues lo que estoy devolviendo es contenido html.

Ver imagen:



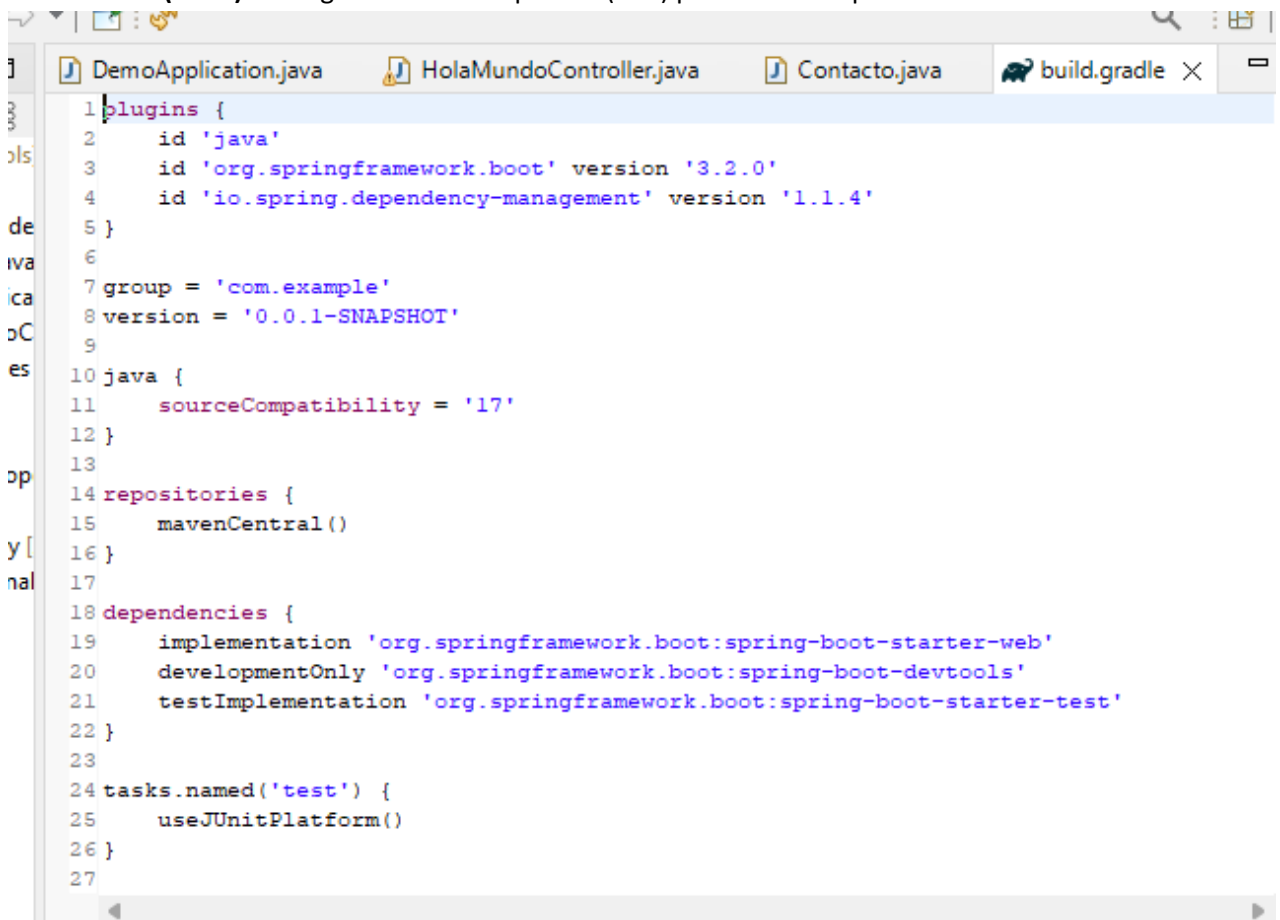
- Echa un vistazo al archivo build.gradle y no dejes de documentar la lectura del código del HelloWorldController.java.

**Este archivo es un archivo de configuración y en este caso es utilizado para definir las dependencias, plugins y configuraciones de nuestro proyecto.**

**-explico mi imagen:**

- Plugins:**
  - java:** habilita la funcionalidad básica de Java para tu proyecto.
  - org.springframework.boot:** proporciona la integración con Spring Boot y permite construir aplicaciones Spring Boot.
  - io.spring.dependency-management:** ayuda a gestionar las versiones de las dependencias.
- group:** Define el grupo del proyecto.
- version:** Define la versión de tu aplicación.

- **Java – sourceCompatibility:** versión de compatibilidad de origen. En este caso, estás utilizando Java 17.
- **Repositories – mavenCentral():** Agrega Maven Central como repositorio de dependencias.
- **Dependencies:**
  - **Implementation:** Agrega las dependencias necesarias para construir una aplicación web con Spring Boot.
  - **developmentOnly:** Agrega las herramientas de desarrollo de Spring Boot, útiles durante el desarrollo.
  - **testImplementation:** Agrega dependencias para realizar pruebas unitarias con Spring Boot.
- **Tasks.named('test'):** Configura la tarea de prueba (test) para utilizar la plataforma JUnit.



```

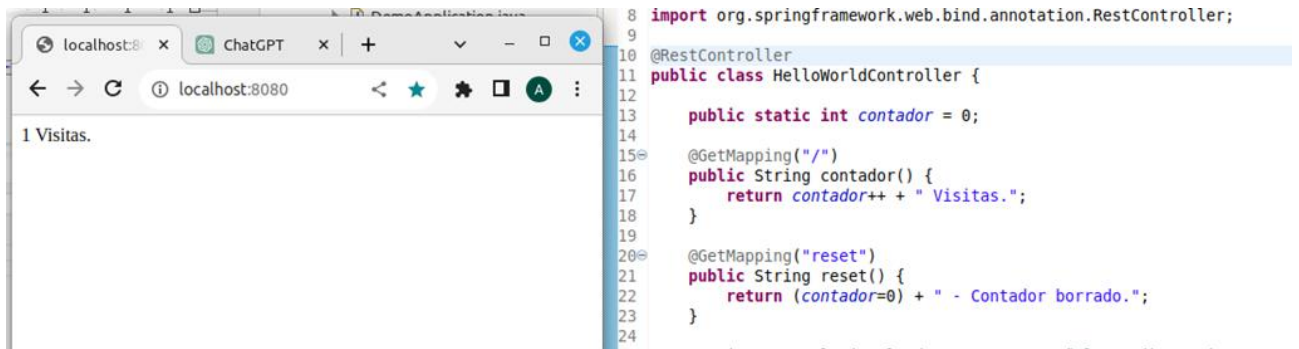
1 plugins {
2     id 'java'
3     id 'org.springframework.boot' version '3.2.0'
4     id 'io.spring.dependency-management' version '1.1.4'
5 }
6
7 group = 'com.example'
8 version = '0.0.1-SNAPSHOT'
9
10 java {
11     sourceCompatibility = '17'
12 }
13
14 repositories {
15     mavenCentral()
16 }
17
18 dependencies {
19     implementation 'org.springframework.boot:spring-boot-starter-web'
20     developmentOnly 'org.springframework.boot:spring-boot-devtools'
21     testImplementation 'org.springframework.boot:spring-boot-starter-test'
22 }
23
24 tasks.named('test') {
25     useJUnitPlatform()
26 }
27
  
```

## 5. CONTADOR DE VISITAS y DIFERENCIAS CON PHP:

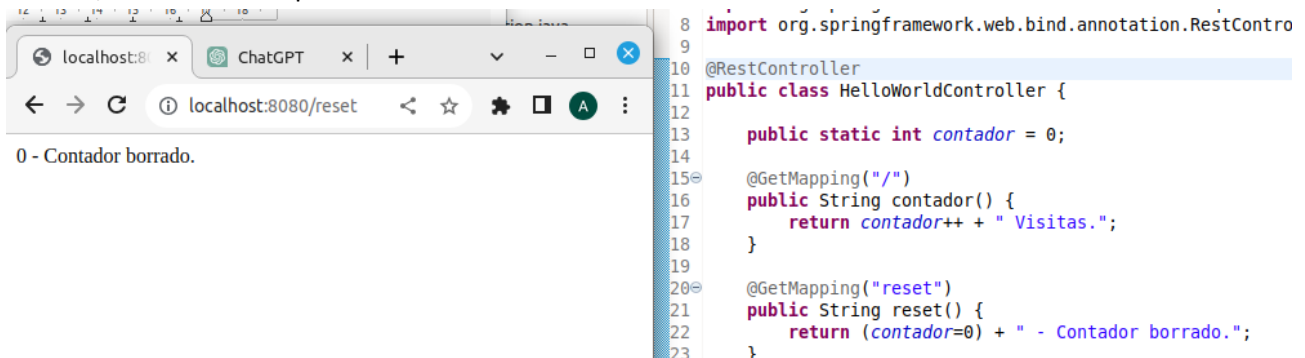
Creo en la misma clase un contador de visitas con variable static a 0.

En GetMapping solo pongo el carácter “/” para que se ejecute cuando solo ponga el puerto, y en return la variable contador con incremento cada vez que se accede.

Y pruebo en el navegador, lo pruebo varias veces y veo que funciona:



También creo un método reset, también con getmapping, que devuelve el contador a 0, para que cuando lo llame, el contador se quede a 0:



### Diferencias más significativas JAVA - PHP:

#### - Persistencia de datos:

- En java podemos usar atributos estáticos para mantener el estado entre las solicitudes pero no entre reinicios de servidor. Si reiniciamos el servidor, el contador vuelve a cero.
- En php, la persistencia de datos entre solicitudes se hace normalmente mediante bbdd o archivos de sesión. Php no mantiene estado de variables entre solicitudes HTTP sin mecanismos externos.

#### - Arquitectura:

- Java, tiende a seguir una arquitectura orientada a objetos.
- Php, tiende a usar un lenguaje de secuencia de comandos y suele ser más flexible y menos estructurada.

### 6. ¿Qué aporta Spring Initializr? ¿Qué aporta Sprint Tools Suite?

#### • SPRING INITIALIZR

Es una herramienta en línea que facilita/aporta la inicialización y rápida configuración de proyectos basados en Spring. Puede generar un esqueleto de proyecto de Spring Boot con las configuraciones básicas y las dependencias que necesitas.

#### • SPRING TOOLS SUITE

Es un IDE basado en Eclipse y diseñado específicamente para el desarrollo de aplicaciones basadas en el framework Spring. Por lo que aporta un entorno de desarrollo completo y optimizado, con herramientas específicas.



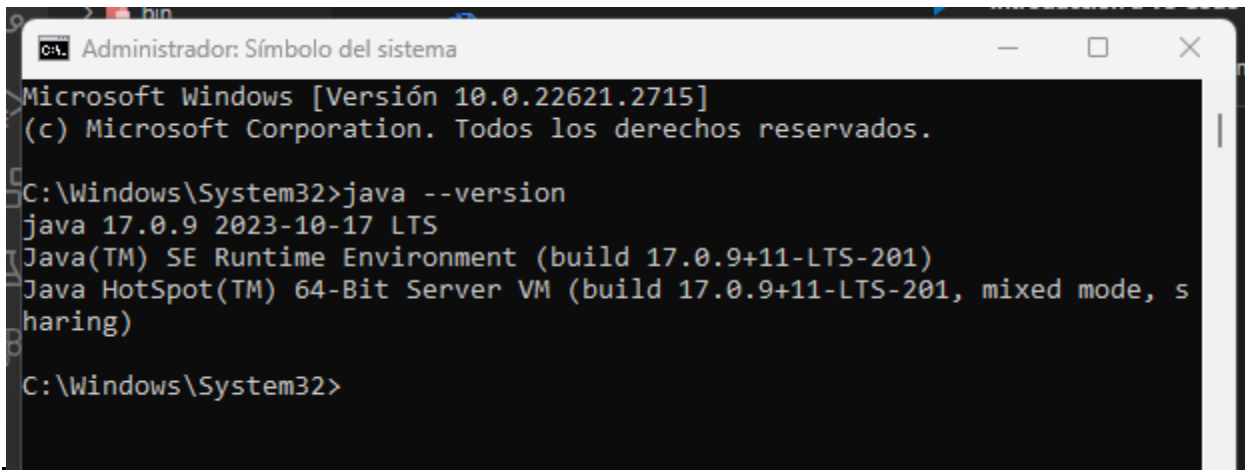
**7. EXPORTAR APLICACIÓN COMO UN JAR:** y muestra cómo se puede ejecutar desde línea de comandos sin IDE (y luego sin gradle) e igualmente levanta su servidor web embebido.

**\*\*MUY IMPORTANTE\*\***

Antes de nada, actualizo mi java a java 17, la misma versión que usa springboot y hago muchas configuraciones siguiendo el tutorial de esta página:

[https://oregoom.com/java/java-development-](https://oregoom.com/java/java-development-kit/#:~:text=Para%20poder%20descargar%20Java%20JDK,instalador%20de%20Java%20JDK%2017.)

[kit/#:~:text=Para%20poder%20descargar%20Java%20JDK,instalador%20de%20Java%20JDK%2017.](https://oregoom.com/java/java-development-kit/#:~:text=Para%20poder%20descargar%20Java%20JDK,instalador%20de%20Java%20JDK%2017.)



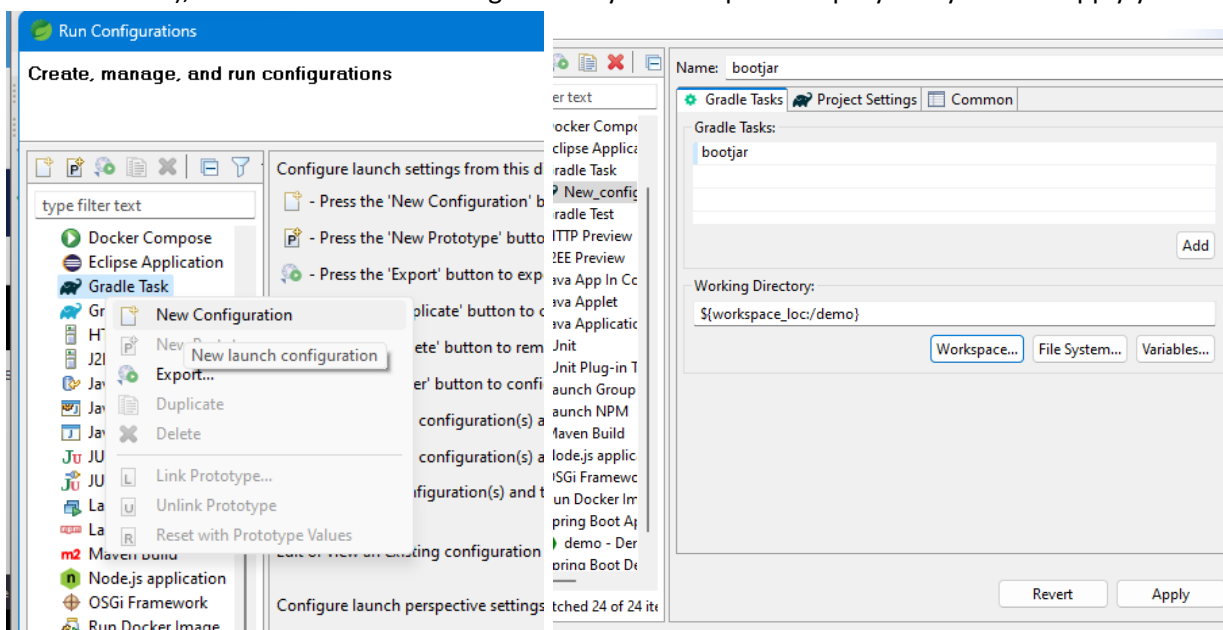
```

C:\Windows\System32>java --version
java 17.0.9 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 17.0.9+11-LTS-201)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.9+11-LTS-201, mixed mode, sharing)

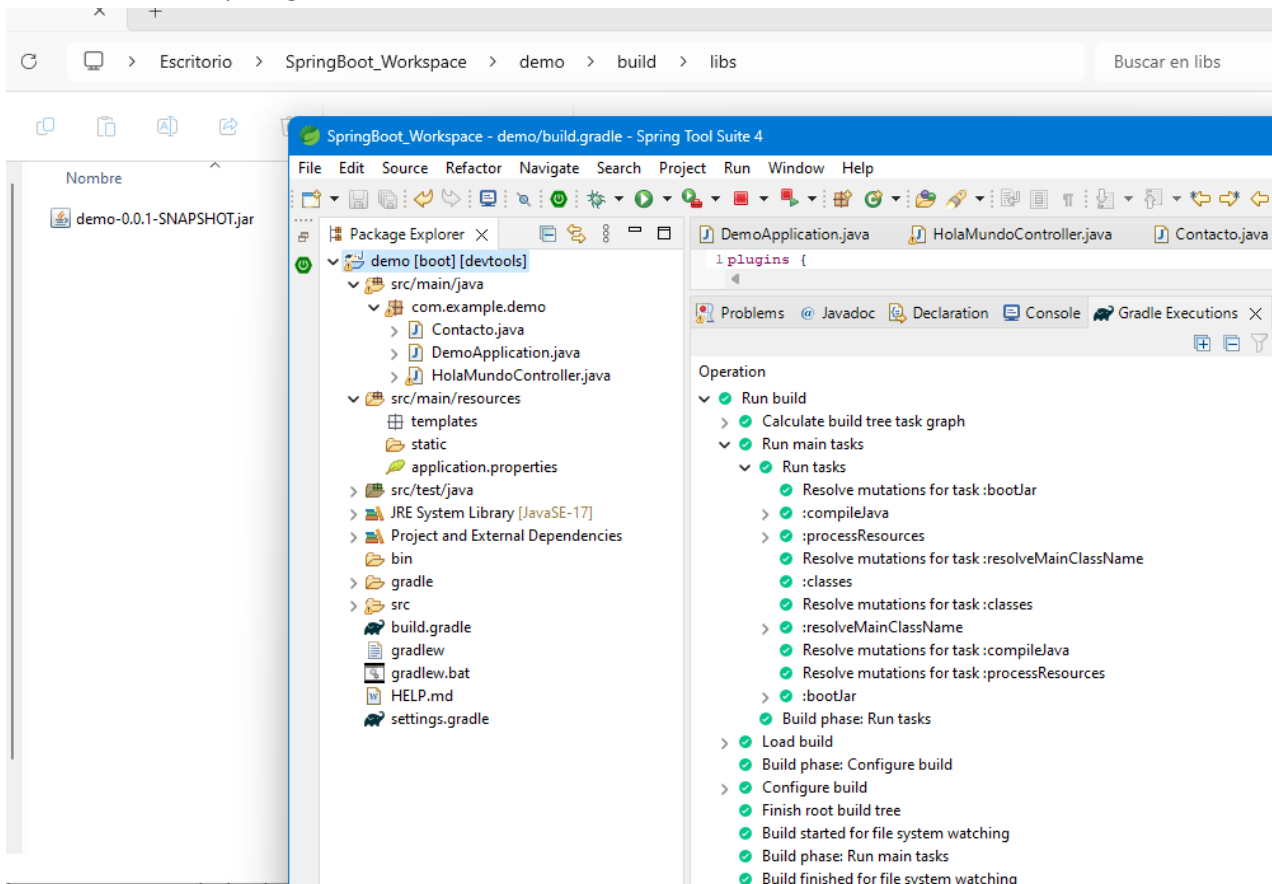
C:\Windows\System32>
  
```

Sigo los pasos indicados:

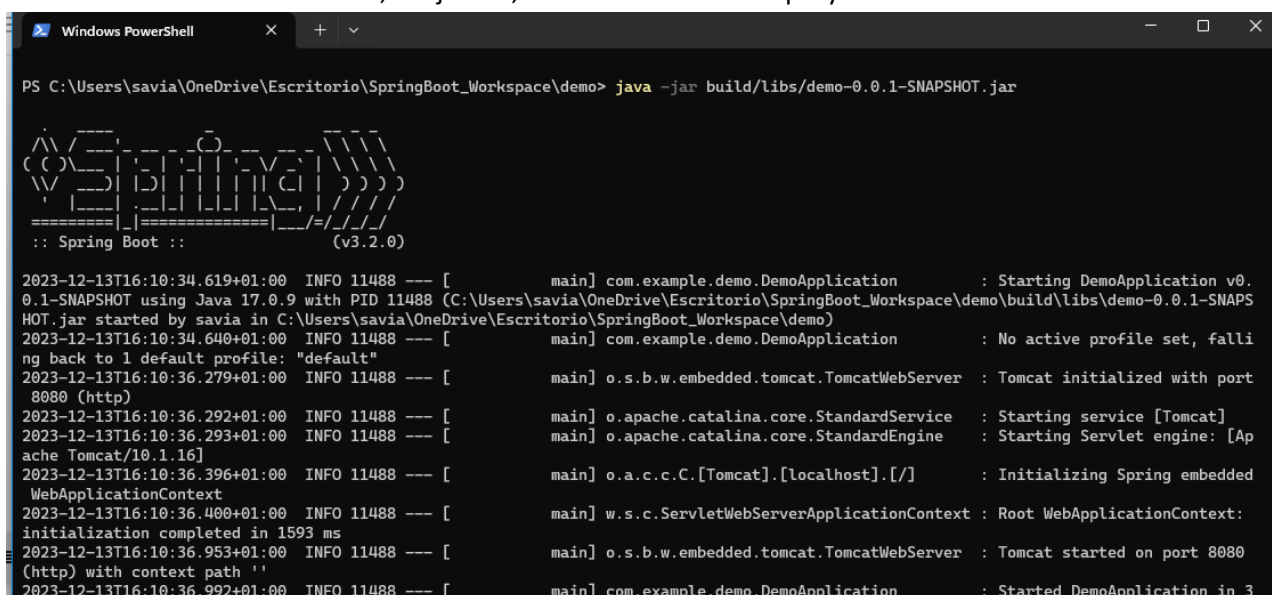
- En el proyecto, botón derecho selecciona Run As → Run Configurations y en la columna izquierda selecciona GradleTask. Encima de Gradle pulsa botón derecho y da a New Configuration.
- Crea una nueva con el nombre bootJar y una sola Gradle Tasks que también sea bootJar (botón añadir), selecciona como Working Directory el workspace del proyecto y al dar a Apply y Run.



- Generará en la carpeta build/libs el jar autoejectuable del proyecto. Lo puedes correr con `java -jar archivo-generado.jar` pero ojo, necesitarás un java con una versión compatible con la que ha usado STS para generarlo.



- Y en línea de comandos, lo ejecuto, estando en la raíz del proyecto:



- Ahora vamos a hacerlo con `./gradlew bootRun` para ejecutarlo sin gradle: usamos los comandos `./gradlew build` y luego `./gradlew bootjar`, y por último con `java -jar` y la ruta de nuestro archivo jar:

[illegible]

- Uso el comando `./gradlew tasks --all` para echar un ojo. Este comando nos muestra una lista de las tareas disponibles del proyecto Gradle, junto con información adicional, como la descripción de cada tarea y a qué plugins pertenecen.

Es útil para explorar las capacidades de tu proyecto Gradle y entender qué tareas están disponibles para su ejecución. También puede ayudarte a identificar las tareas específicas proporcionadas por los plugins que has aplicado en tu proyecto.

```

PS C:\Users\savia\OneDrive\Escritorio\SpringBoot_Workspace\demo> ./gradlew tasks --all

> Task :tasks

-----
Tasks runnable from root project 'demo'
-----

Application tasks
-----
bootRun - Runs this project as a Spring Boot application.
bootTestRun - Runs this project as a Spring Boot application using the test runtime classpath.

Build tasks
-----
assemble - Assembles the outputs of this project.
bootBuildImage - Builds an OCI image of the application using the output of the bootJar task
bootJar - Assembles an executable jar archive containing the main classes and their dependencies.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the classes of the 'main' feature.
resolveMainClassName - Resolves the name of the application's main class.
resolveTestMainClassName - Resolves the name of the application's test main class.
testClasses - Assembles test classes.

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Documentation tasks
-----
javadoc - Generates Javadoc API documentation for the 'main' feature.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'demo'.
dependencies - Displays all dependencies declared in root project 'demo'.
dependencyInsight - Displays the insight into a specific dependency in root project 'demo'.
dependencyManagement - Displays the dependency management declared in root project 'demo'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'demo'.
projects - Displays the sub-projects of root project 'demo'.
properties - Displays the properties of root project 'demo'.
resolvableConfigurations - Displays the configurations that can be resolved in root project 'demo'.
tasks - Displays the tasks runnable from root project 'demo'.

Verification tasks
-----
check - Runs all checks.
test - Runs the test suite.

Other tasks
-----
compileJava - Compiles main Java source.
compileTestJava - Compiles test Java source.
components - Displays the components produced by root project 'demo'. [deprecated]
dependentComponents - Displays the dependent components of components in root project 'demo'. [deprecated]
model - Displays the configuration model of root project 'demo'. [deprecated]
prepareKotlinBuildScriptModel
processResources - Processes main resources.
processTestResources - Processes test resources.

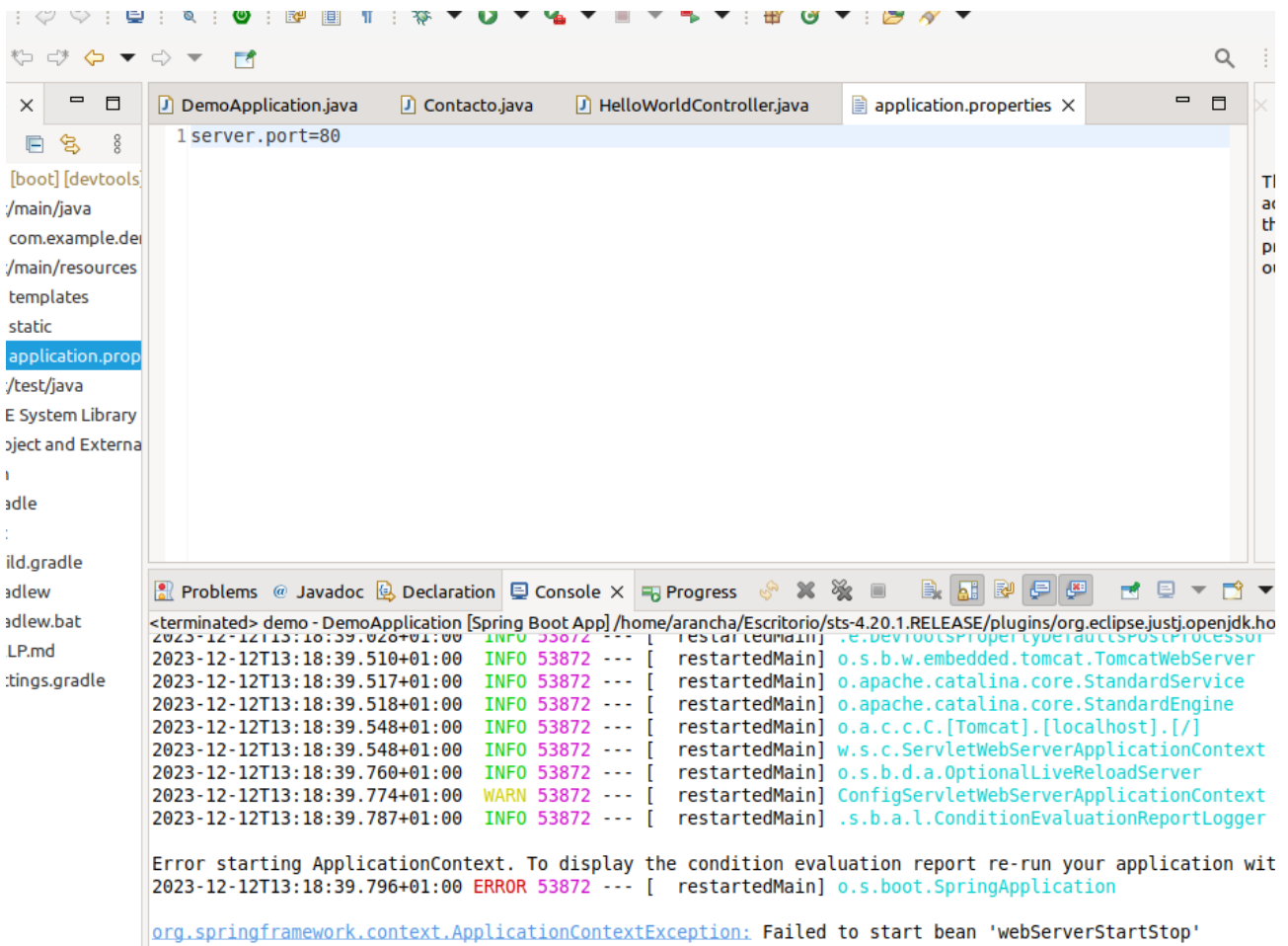
Rules

Rules
Pattern: clean<TaskName>: Cleans the output files of a task.
Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.

```

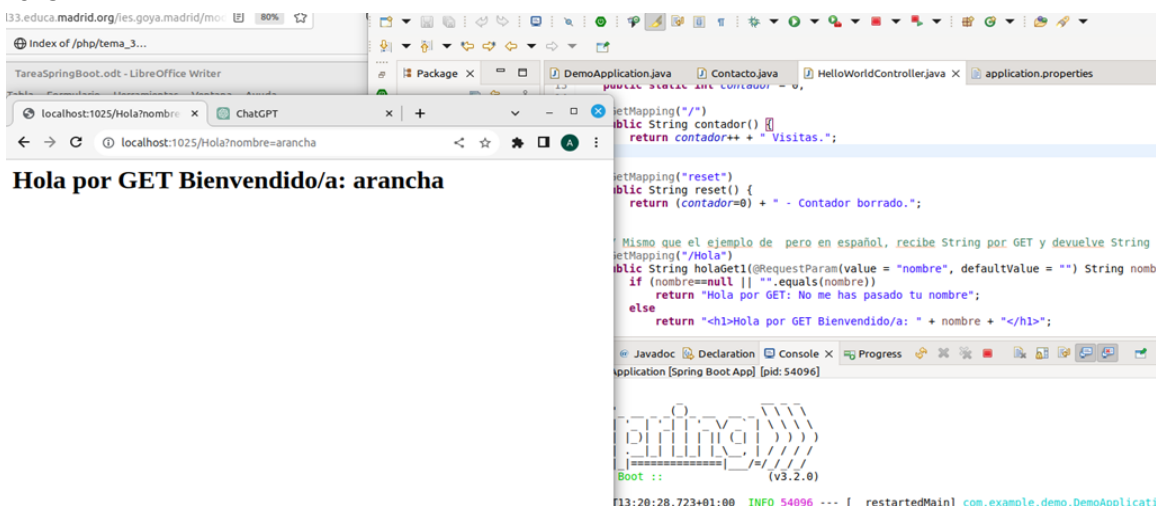
**Avanzado:** ¿Puedes cambiar el puerto en el que atiende la aplicación? ¿Puedes poner e puerto 80?

**Para cambiar el puerto hay que modificar el archivo `application.properties`, localizado en `src/main/resources`, añadiendo una línea `server.port=x`, siendo x el número de puerto que queremos usar.**



Pero con el puerto 80 da error pues hay que poner un puerto por encima de 1024 porque los puertos por debajo del número 1024 están reservados para servicios que requieren permisos de superusuario (root) para ejecutarse.

Por lo que usamos el puerto 1025 y probamos en el navegador con el método Hola de GET e indicando el 1025:



## CONCLUSIONES

Con las indicaciones descritas y ejemplos realizados por el profesor no ha sido difícil hacer la práctica, la guía para hacer el punto 7 estaba muy clara. Tal vez lo más complejo ha sido actualizar a la versión JAVA 17, pues no es solo descargar e instalar, hay que hacer muchas configuraciones en el sistema, pero gracias al tutorial encontrado, lo he podido seguir bien (link indicado en el punto 7).

También en el punto 4, al empezar al hacer las pruebas desde el navegador, al inicio no me funcionaba pero el error era fácil de solucionar, solo tenía que añadir la línea `@RestController` (mismo fallo lo tuve en clase).

Ha sido un poco “intenso” inicialmente porque ha sido un cambio radical a lo que hemos visto anteriormente, pero me ha gustado usar un poco de código desde el IDE y empezar a usar Springboot para volver a ver código Java. De hecho, he realizado el punto 4 tres veces, en clase antes del puente, en clase después del puente y en casa.

En mi caso no le he dedicado mucho tiempo, a parte del dedicado en clase, en casa solo he estado dos días. Pues los días de puente no he podido tener acceso. Por lo que es una muestra de que ha sido una práctica muy asequible.