

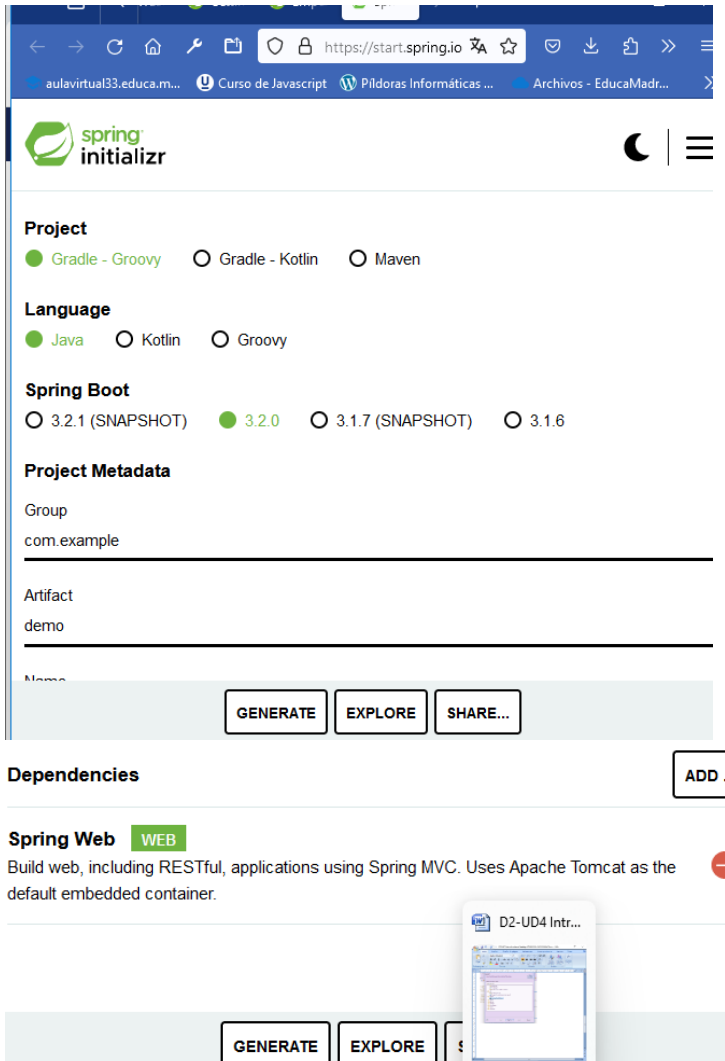
Tarea 2 – UD4

Introducción al testing

TUTORIAL: <https://spring.io/guides/gs/testing-web/>

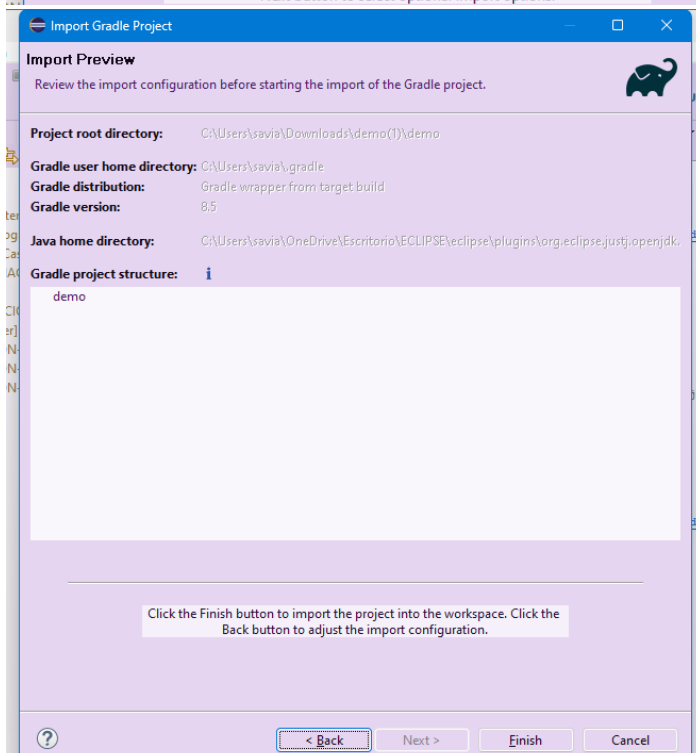
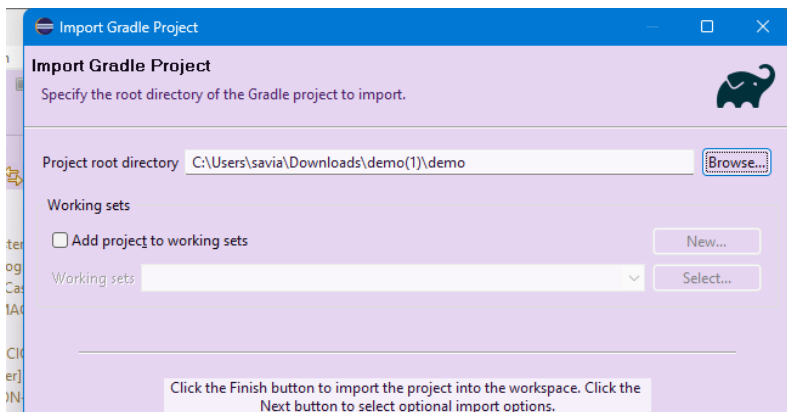
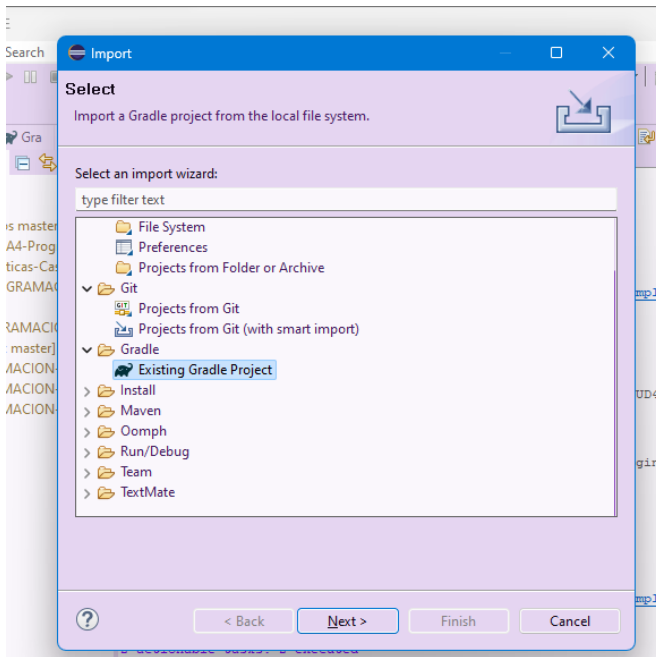
Voy a usar Eclipse.

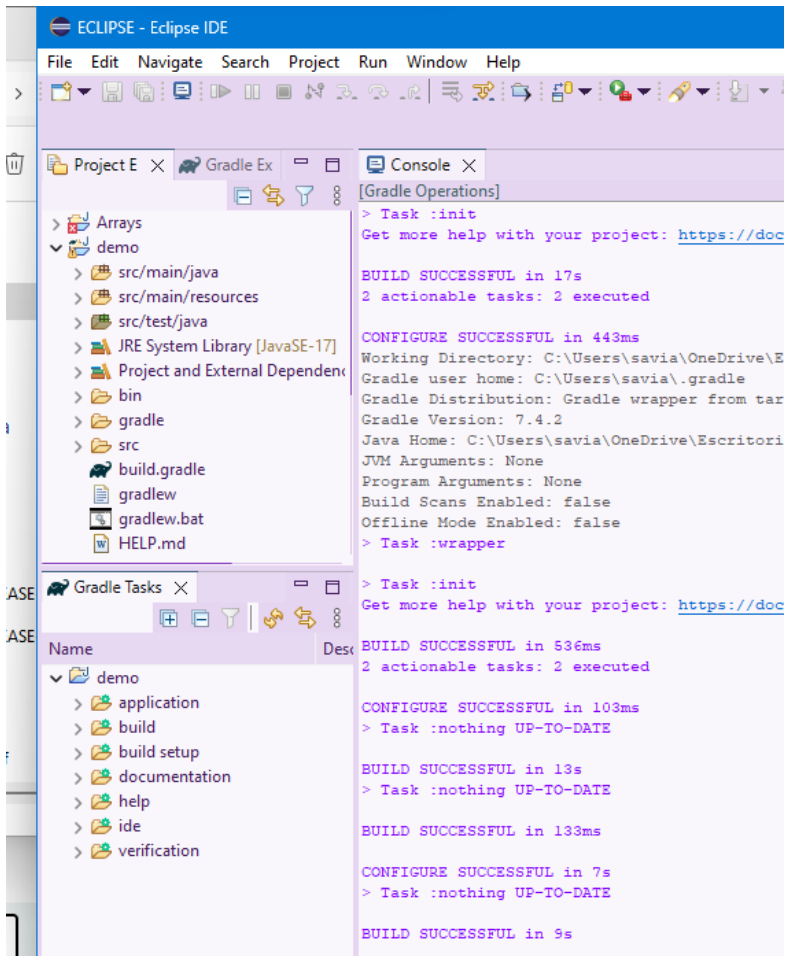
Genero zip de proyecto gradle, seleccionando SpringWeb en dependencias y java como lenguaje:



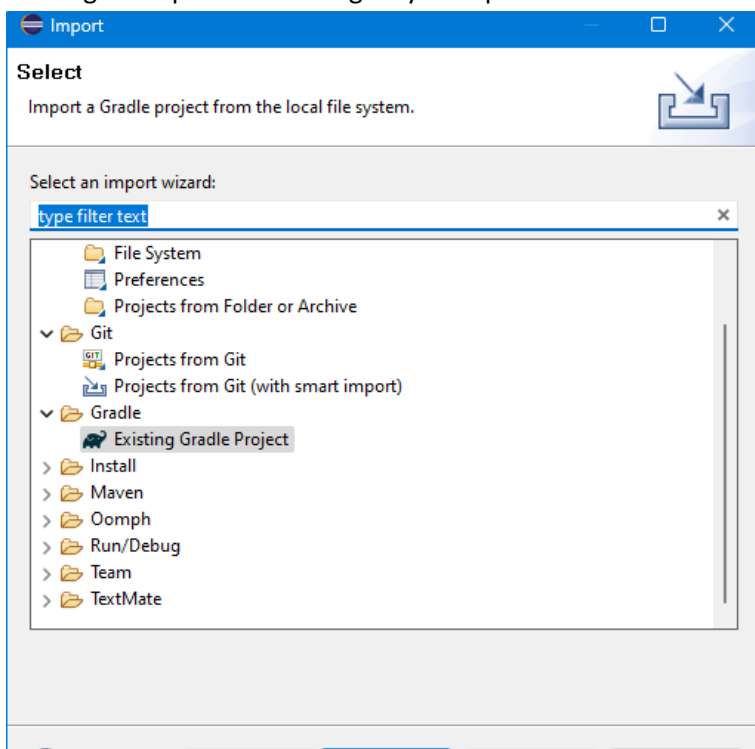
The screenshot shows the Spring Initializr web application interface. The browser address bar displays <https://start.spring.io>. The page features the Spring logo and a navigation menu. Under the 'Project' section, 'Gradle - Groovy' is selected. The 'Language' section has 'Java' selected. Under 'Spring Boot', version '3.2.0' is selected. The 'Project Metadata' section includes input fields for 'Group' (containing 'com.example') and 'Artifact' (containing 'demo'). At the bottom of this section are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. Below this is the 'Dependencies' section, which includes an 'ADD ..' button. A dependency named 'Spring Web' is listed with a 'WEB' tag and a description: 'Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.' Below the dependencies are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. A small thumbnail image of a document titled 'D2-UD4 Intr...' is visible in the bottom right corner of the screenshot.

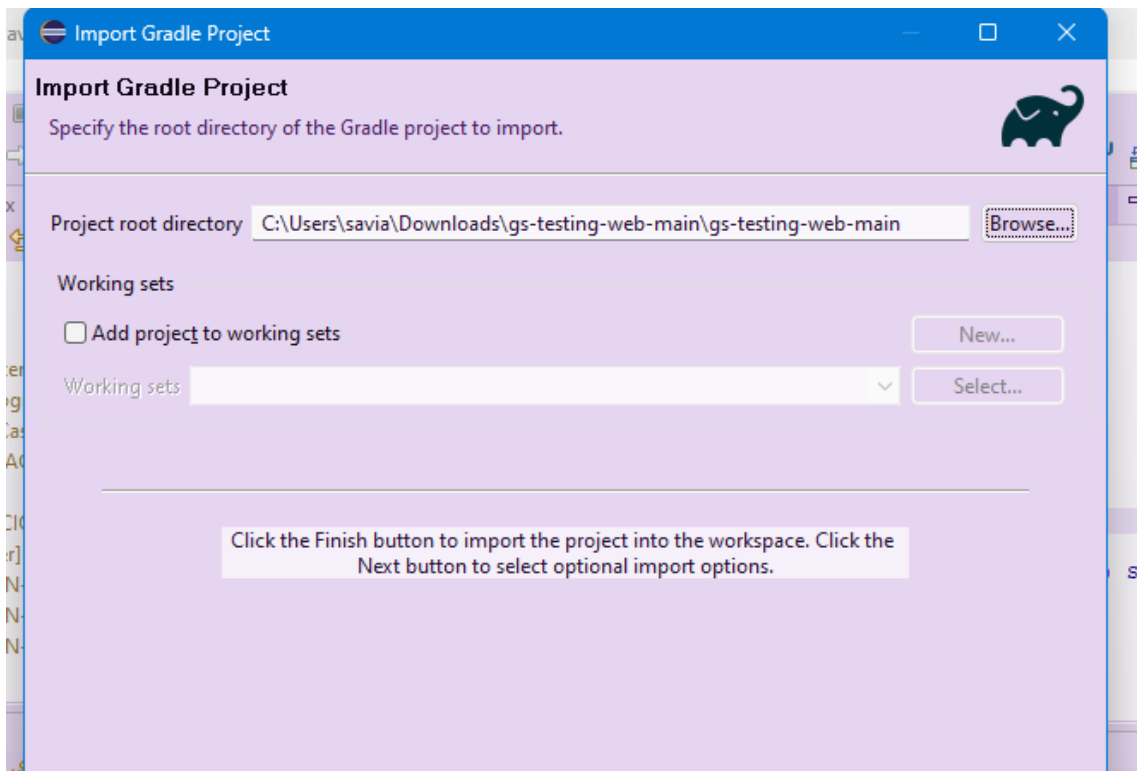
Y lo importo en Eclipse como Gradle project:



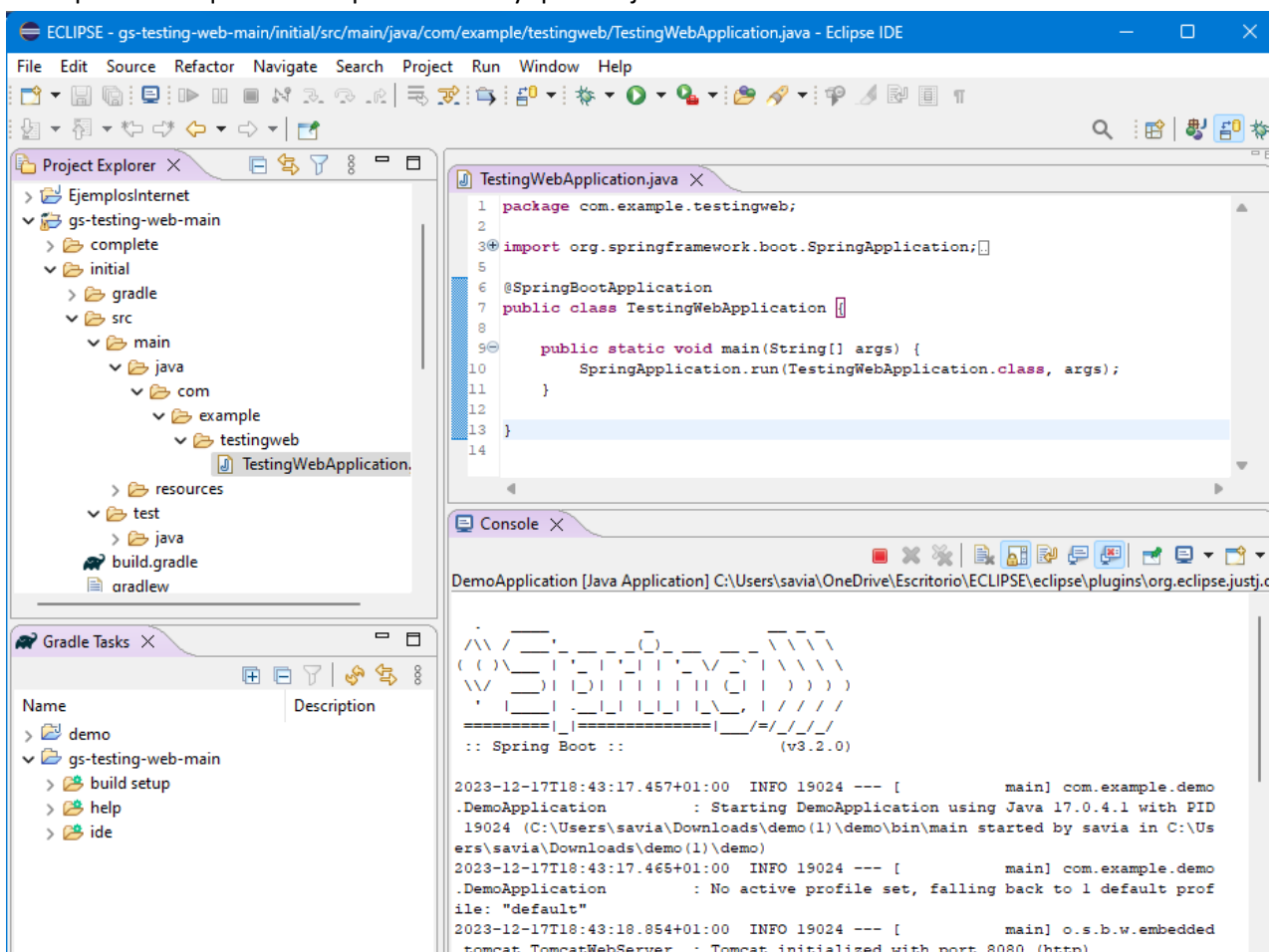


Descargo el repositorio de la guía y lo importo como Gradle Project:



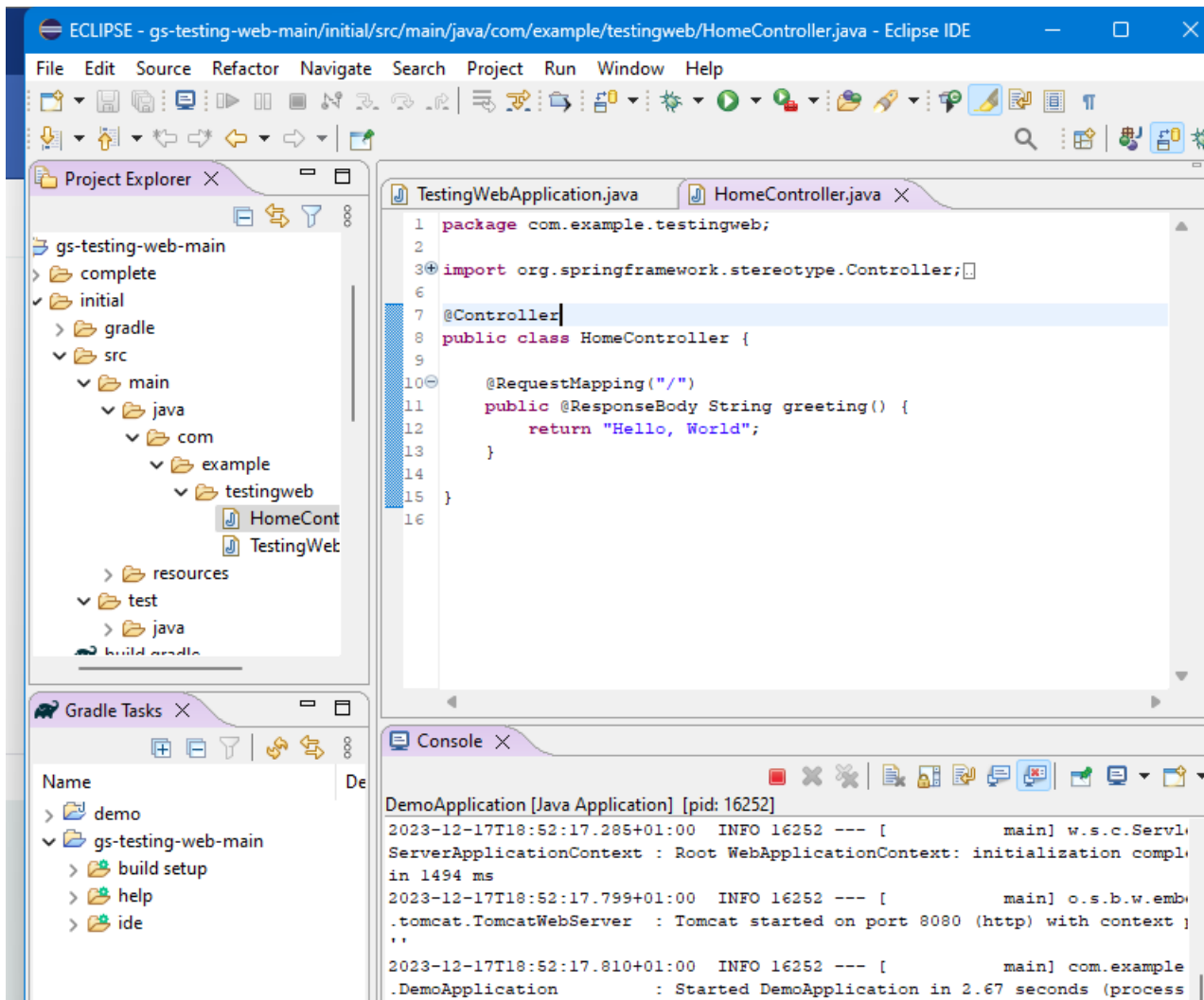


Y comprobamos que se ha importado bien y que se ejecuta:

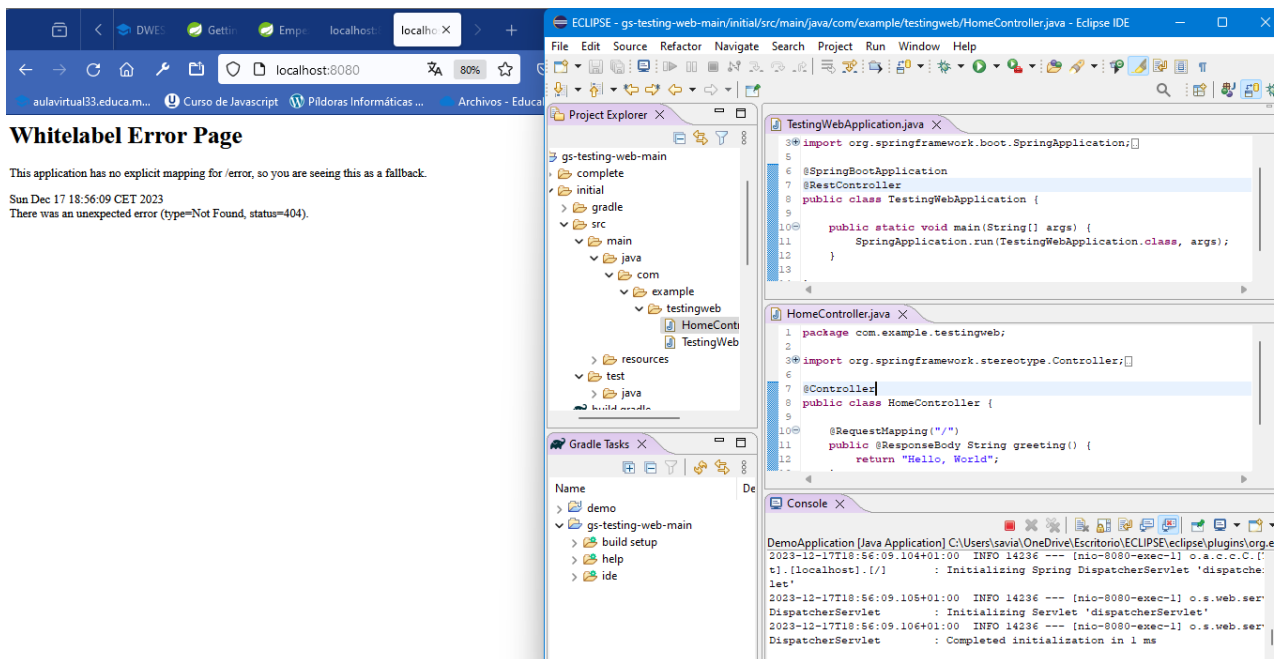


Crear una aplicación simple:

Creamos nuevo controlador en la clase HomeController:

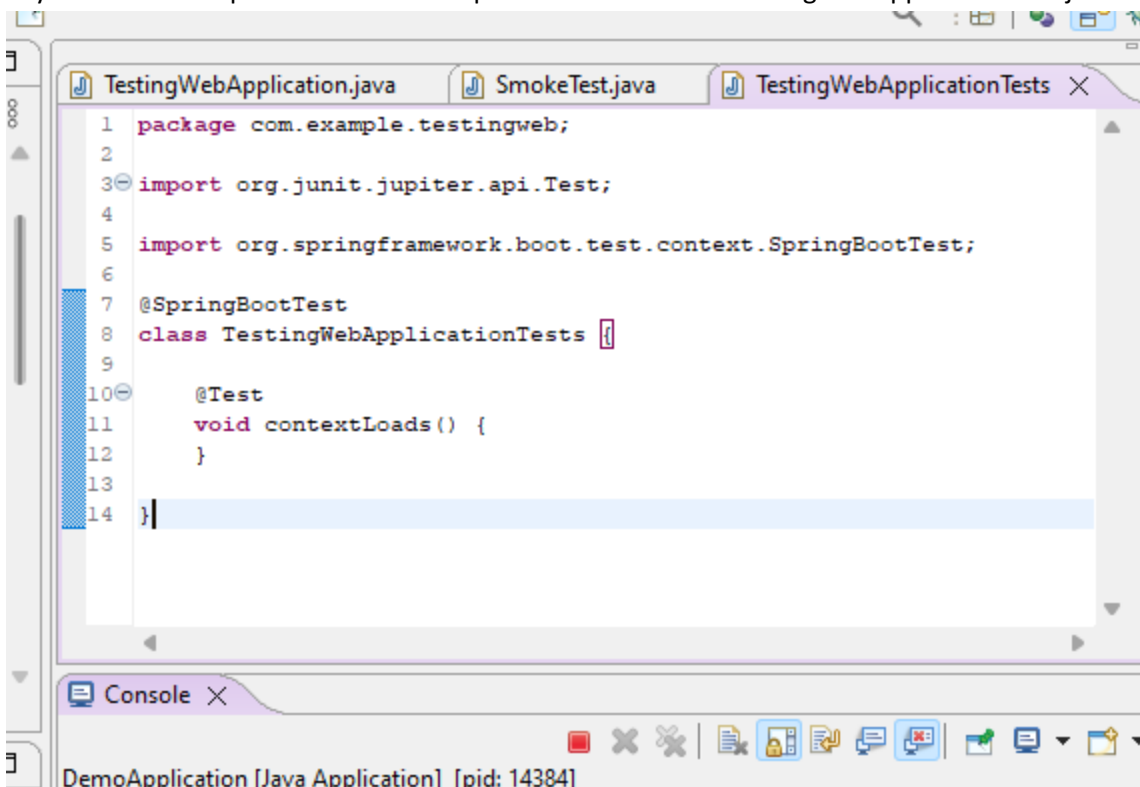


Ahora probamos a carga la página de inicio en el navegador, pero no sale nada, aunque en la consola de eclipse no sale ningún error:



Aún así, sigo con el tutorial:

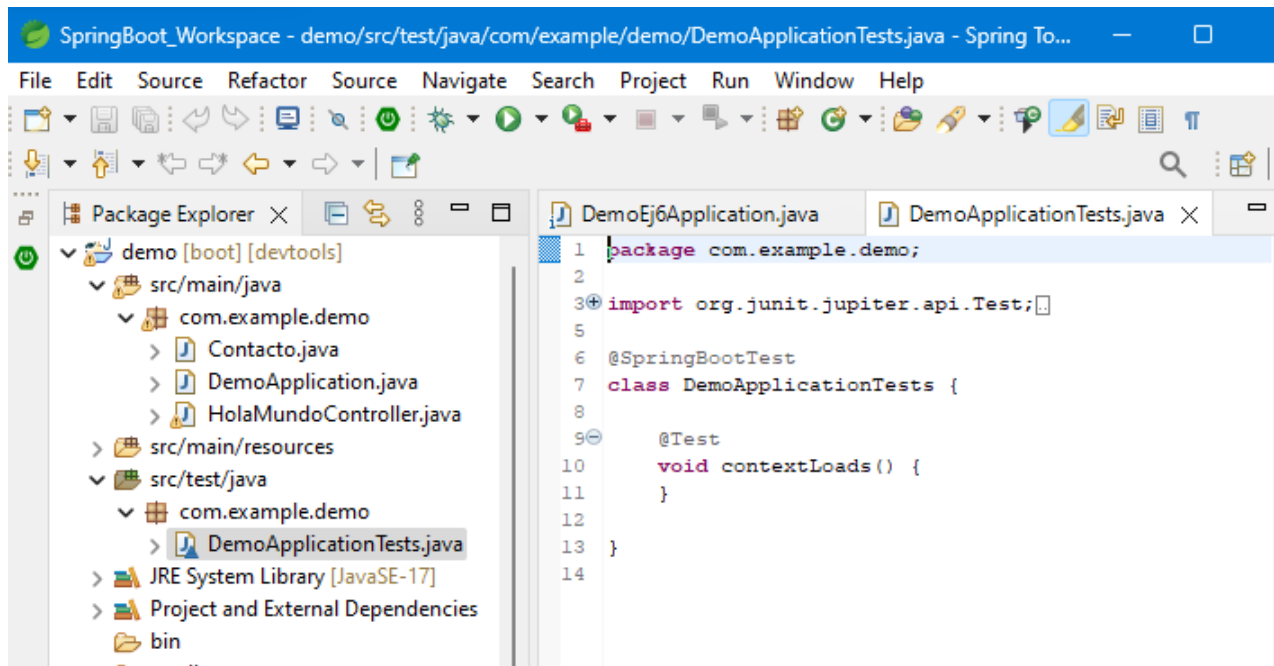
Voy a automatizar la prueba. Escribo una prueba de control en TestingWebApplicationTest.java:



Aquí empecé a tener problemas por lo que voy al paso avanzado usando el código del desafío anterior:

1. Salta el tutorial hasta el punto "Run the application" pues la parte anterior es lo ya hecho en el desafío anterior.
2. Mira si el proyecto ha creado algo en src/test/java. Si estuviera una clase tipo DemoApplicationTests o similar creada automáticamente en tu proyecto. Aprovecha para leer el significado de las

anotaciones en el tutorial, aunque algunas todavía no las veas. Así es, hay un archivo, lo abro y veo sus anotaciones:



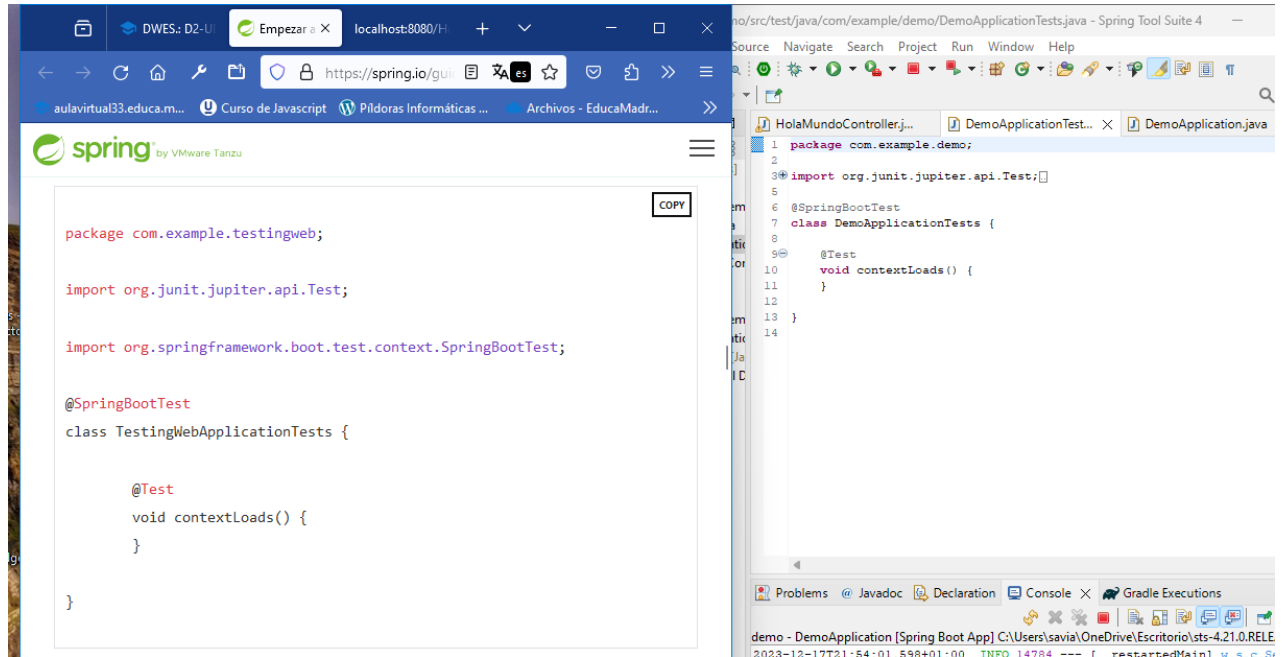
- **@SpringBootTest**
 - indica que esta clase debe cargar la configuración de la aplicación Spring Boot.
 - Cuando ejecutas pruebas con @SpringBootTest, Spring Boot configura el contexto de la aplicación y carga todas las configuraciones necesarias para ejecutar las pruebas.
- **class DemoApplicationTests:**
 - Esta clase representa la suite de pruebas para tu aplicación.
 - Las pruebas específicas se escribirán como métodos dentro de esta clase.
- **@Test:**
 - indica que el método siguiente es una prueba.
 - El nombre del método sugiere que la prueba verifica si el contexto de la aplicación se carga correctamente.
- **void contextLoads() { }:**
 - cuerpo de la prueba. El método está vacío, lo que significa que no contiene ninguna lógica de prueba específica.
 - La presencia de este método generalmente se debe a que, en la fase de desarrollo inicial, simplemente se verifica que el contexto de la aplicación se cargue correctamente.

Cuando ejecutas esta prueba (puedes hacerlo desde tu IDE o utilizando herramientas de construcción como Maven o Gradle), Spring Boot cargará el contexto de la aplicación y verificará que no haya errores de configuración. La prueba contextLoads suele ser la primera prueba que se ejecuta para asegurarse de que la aplicación se esté configurando correctamente.

Puedes agregar pruebas adicionales a esta clase a medida que desarrollas tu aplicación para verificar el comportamiento específico de los componentes y servicios

3. En el apartado “Test the application” compara si el primer código (TestingWebApplicationTest) a parte del nombre, su contenido es igual o no a la clase del punto anterior o en caso contrario créala.

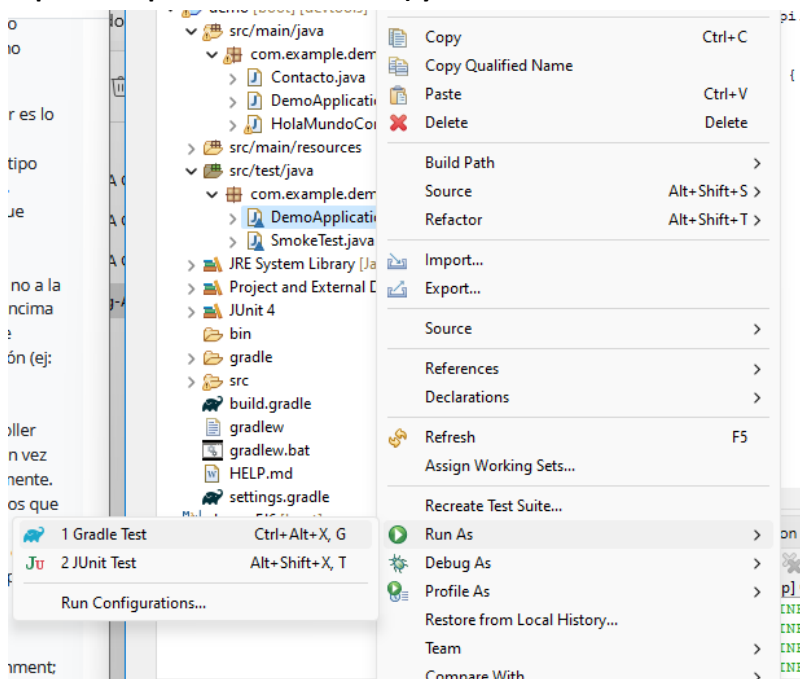
Vemos que es igual:



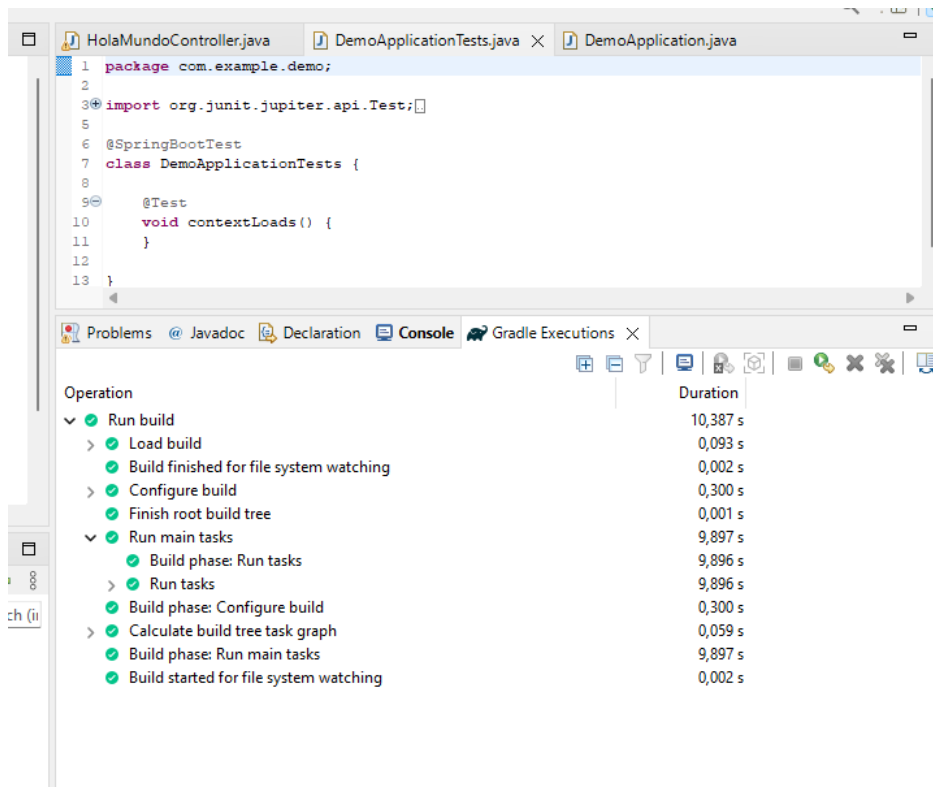
El `@SpringBootTest` anotación le dice a Spring Boot que busque una clase de configuración principal (una con `@SpringBootApplication`, por ejemplo) y utilizarlo para iniciar un contexto de aplicación de Spring.

Con un clic derecho encima de la clase dale a Run → Gradle Test y observa el resultado.

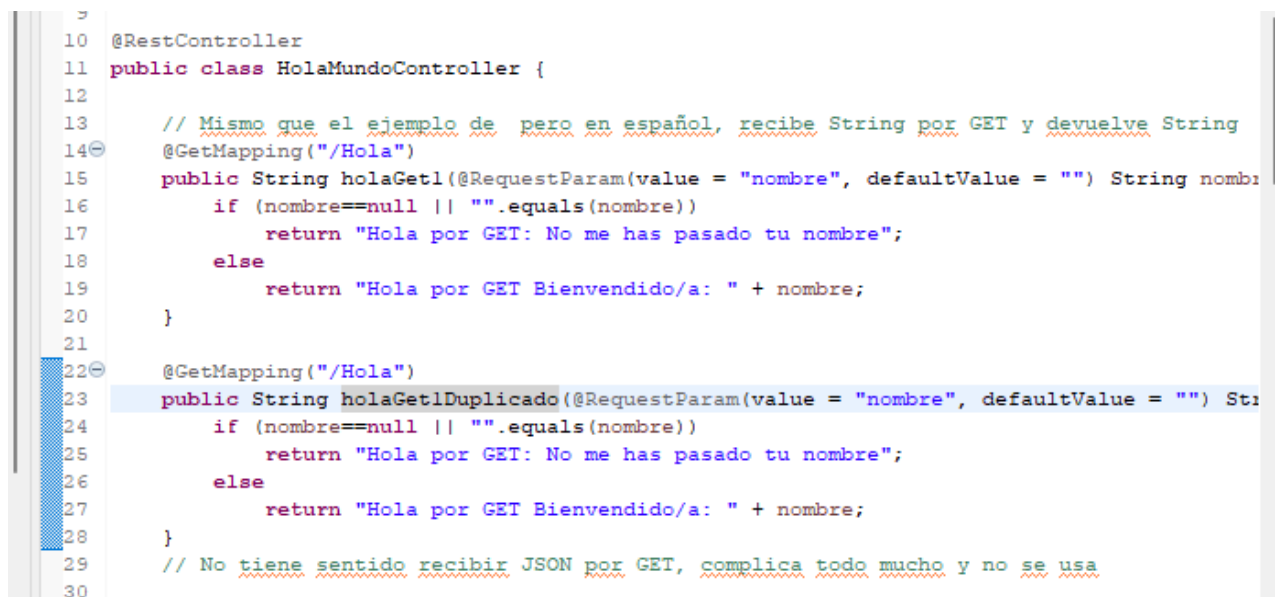
Simplemente comprueba que la aplicación se levanta. Introduce un error en la aplicación (ej: duplicar un punto de acceso web) y vuelve a correr el test ¿cambia algo?



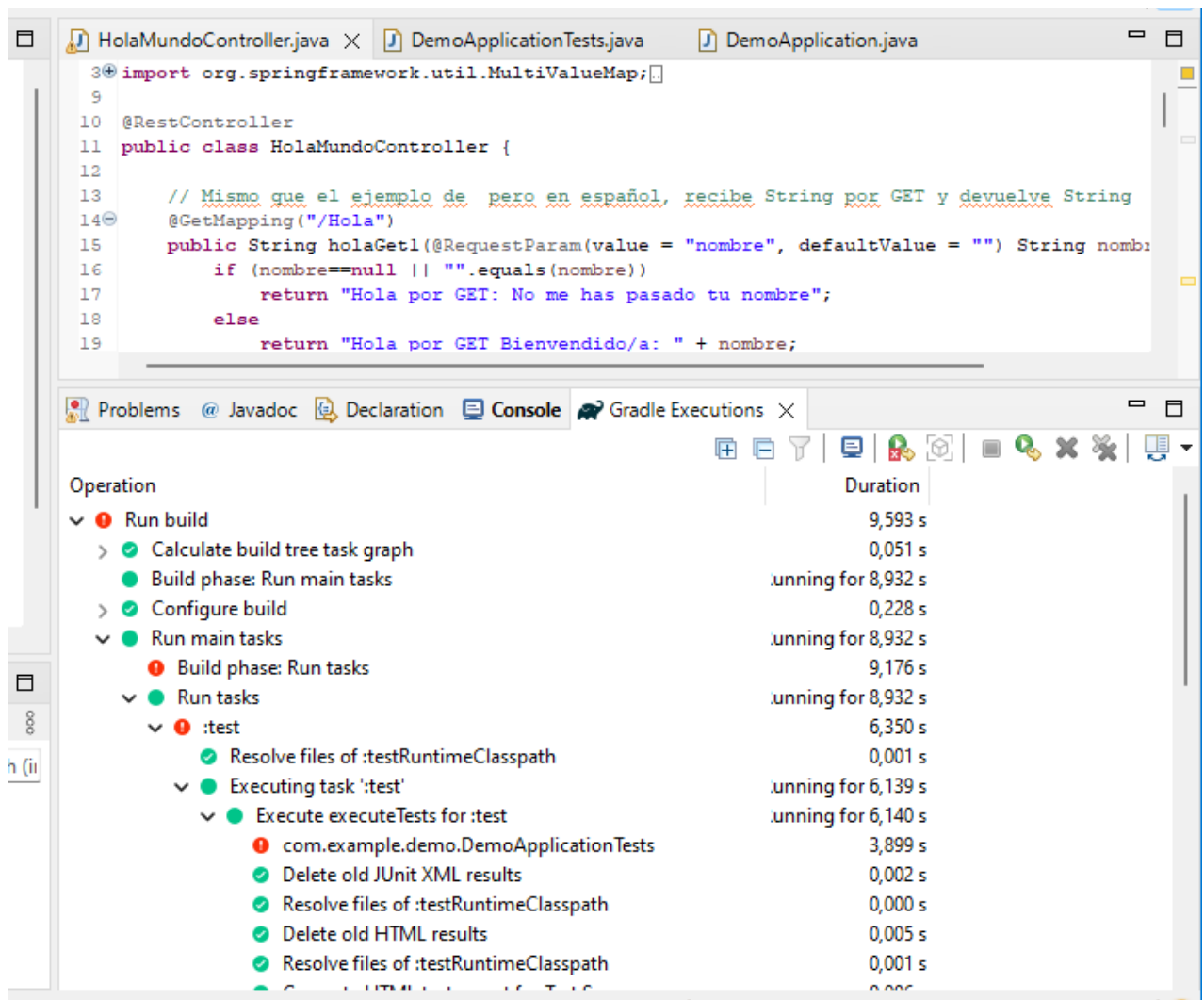
Vemos que todo es correcto, checks verdes:



Ahora vamos a provocar un error en el código, vamos a duplicar el método `holaGet1` para así tener dos métodos con la misma ruta:

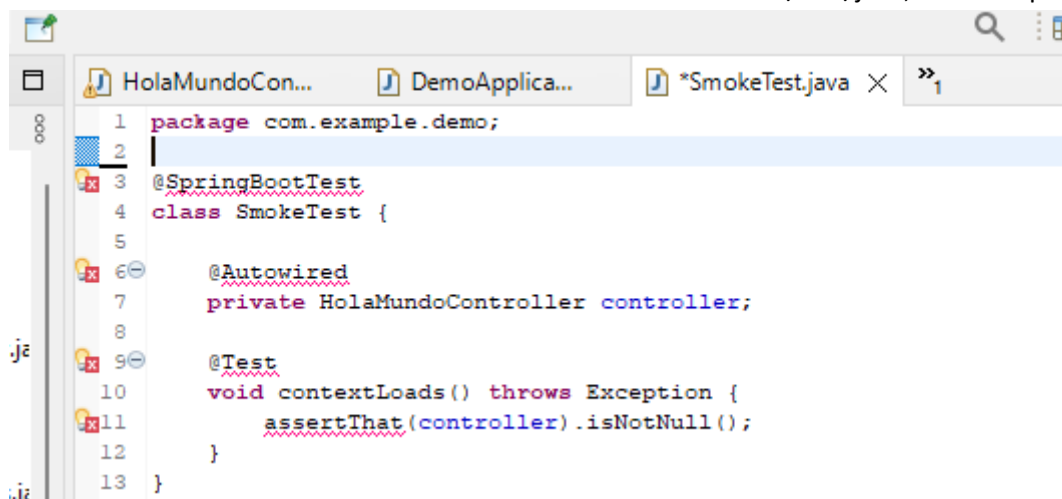


Y al volver a darle a run as Gradle test vemos que hay errores (también salían los errores en la consola):

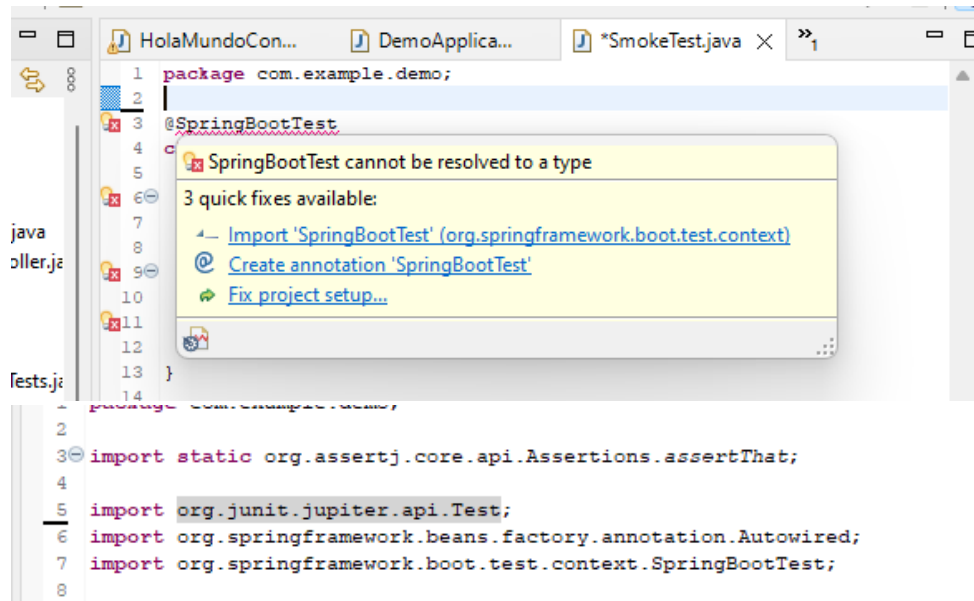


4. Copia el segundo código de ese apartado del tutorial (SmokeTest) sin los imports a una clase dentro de `src/test/java` pero cambiando `HomeController` por `HolaMundoController`, hace algo similar a lo anterior, pero esta vez en vez de la aplicación comprueba que el controlador se ha levantado correctamente. Pruébalo y documenta el resultado. Documenta las anotaciones y métodos que te han pedido incluir nuevos imports. Analiza el código.

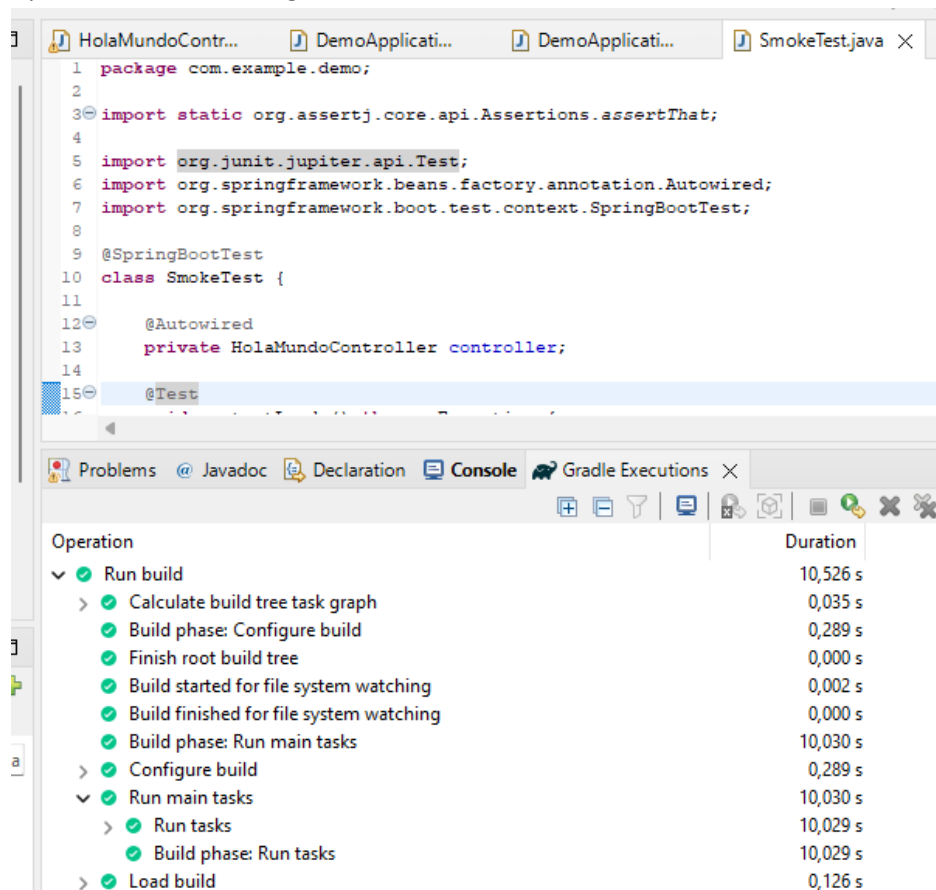
Vamos a añadir una afirmación creando la clase `SmokeTest` en `src/test/java`, sin los imports:



Ahora importamos los imports solicitados para que no salten los errores:



Y probamos con run as gradle test:



Spring interpreta la anotación @Autowired y el controlador se inyecta antes de ejecutar los métodos de test. No veo diferencias de los resultados con el test del código anterior. Solo que este código de pruebas está destinado a verificar que el contexto de la aplicación HolaMundoController carga/inyecta correctamente y el código de DemoApplicationTests es un test vacío que lo estamos usando por defecto, y en un futuro puede usarse para agregar pruebas más específicas.

Ahora voy a documentar el código, las anotaciones y métodos:

- **@SpringBootTest** anotación: indica que este caso de prueba debe cargar la configuración de la aplicación completa.
- **@Autowired** anotación: se utiliza para inyectar `HolaMundoController` en el campo **private HolaMundoController controller** lo que significa que Spring Boot se encargará de crear una instancia del controlador y asignarla al campo `controller` antes de ejecutar el método de prueba.
- **@Test** anotación: indica que el método anotado es un método de prueba. Este método será ejecutado cuando se ejecute la prueba.
- **void contextLoads() throws Exception { ... } método:** método de prueba se que está destinado a verificar la carga del contexto de la aplicación. Está marcado con **throws Exception**, para manejar excepciones que puedan ocurrir durante la ejecución del test.
- **assertThat(controller).isNotNull();** Utiliza la biblioteca AssertJ para realizar una aserción que verifica que el controlador (`controller`) no sea nulo. Si el controlador es nulo, la aserción fallará y la prueba se considerará como fallida.

Este código está probando que `HolaMundoController` se carga correctamente en el contexto de la aplicación de Spring Boot y que no es nulo. Este tipo de prueba se utiliza para verificar rápidamente que la aplicación puede arrancar y que algunos de sus componentes clave están disponibles.

5. Vamos con el tercer código de ese apartado, `HttpRequestTest`, de nuevo copia la clase sin los imports (en clase explicaré por qué) Si necesitarás este import que no es capaz de localizar: `import org.springframework.boot.test.context.SpringBootTest.WebEnvironment`

```

1 package com.example.demo;
2
3 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
4 class HttpRequestTest {
5
6     @LocalServerPort
7     private int port;
8
9     @Autowired
10    private TestRestTemplate restTemplate;
11
12    @Test
13    void greetingShouldReturnDefaultMessage() throws Exception {
14        assertThat(this.restTemplate.getForObject("http://localhost:" +
15            port, String.class)).contains("Hello, World");
16    }
17 }
18

```

E importo los imports:

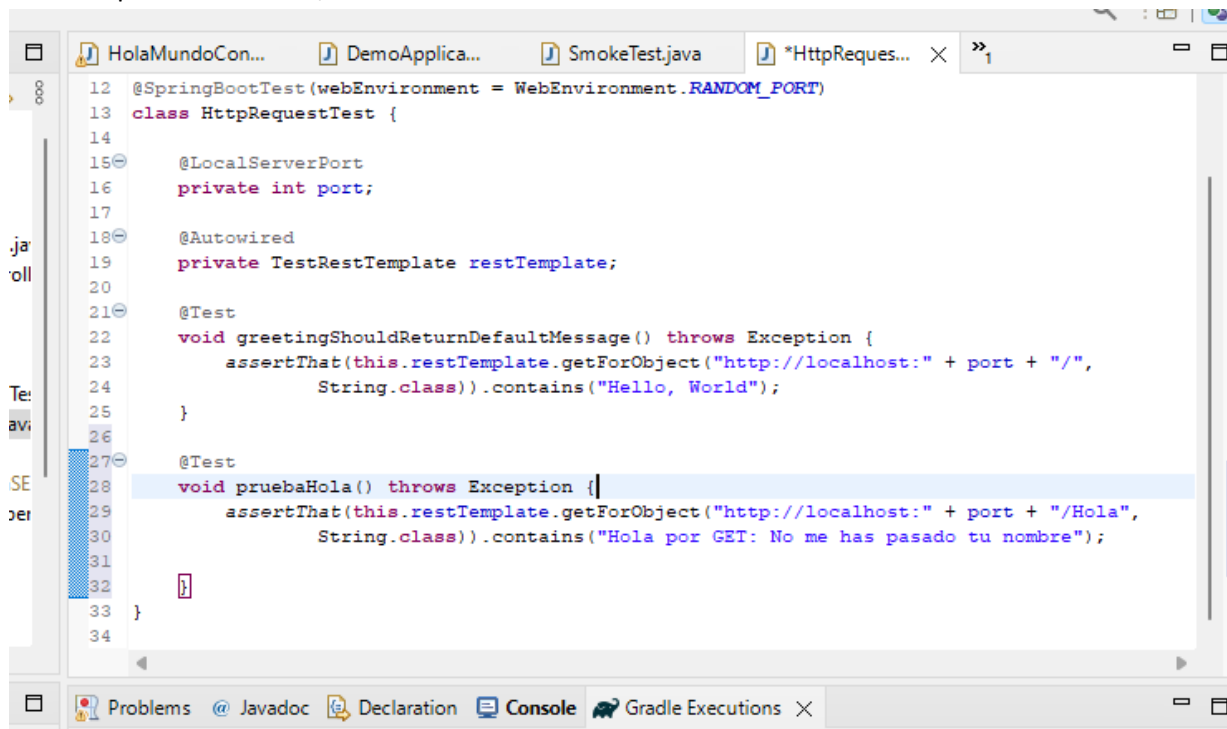
```

3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
9 import org.springframework.boot.test.web.client.TestRestTemplate;
10 import org.springframework.boot.test.web.server.LocalServerPort;
11

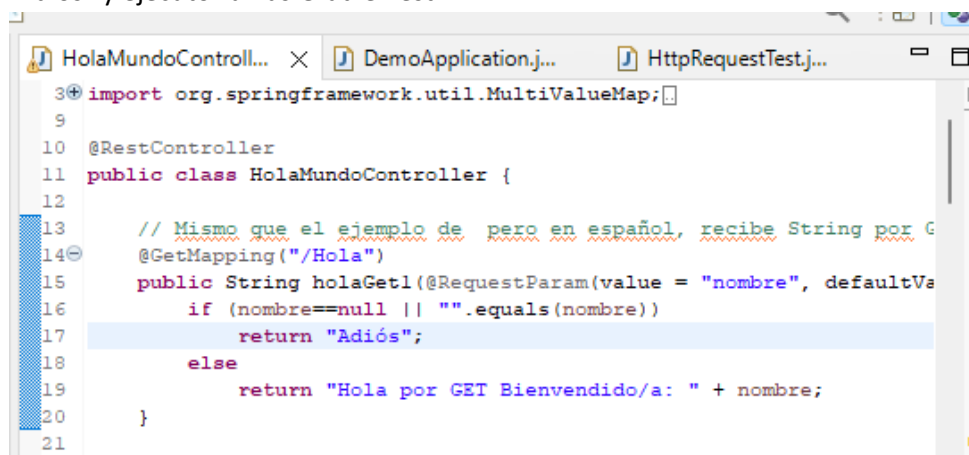
```

Añade el punto de acceso (/Hola) y el valor esperado (Hola por GET: No me has pasado tu nombre) y pruébalo con esos datos y con otros para ver cuándo falla. Localiza el fallo ¿se entiende? Prueba con Run → Junit Test ¿mejor?

Añado el punto de acceso /Hola:



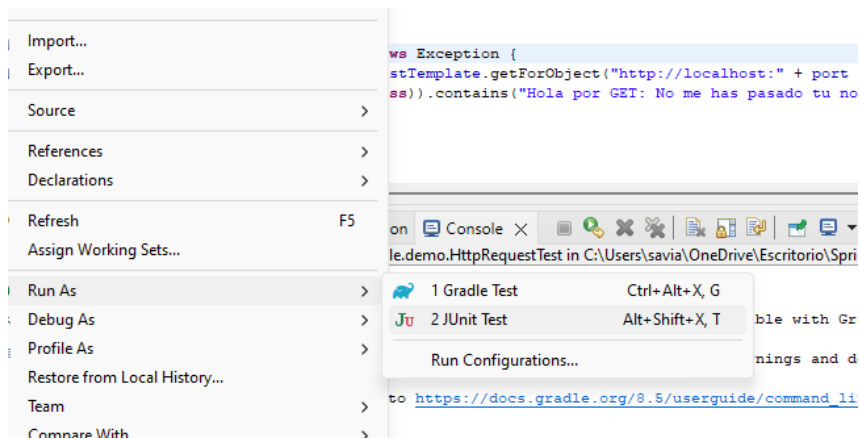
Y en la clase HolaMundoController provocho error en el getMappint /Hola, en el primer return ponemos “Adiós” y ejecuto run as Gradle Test:



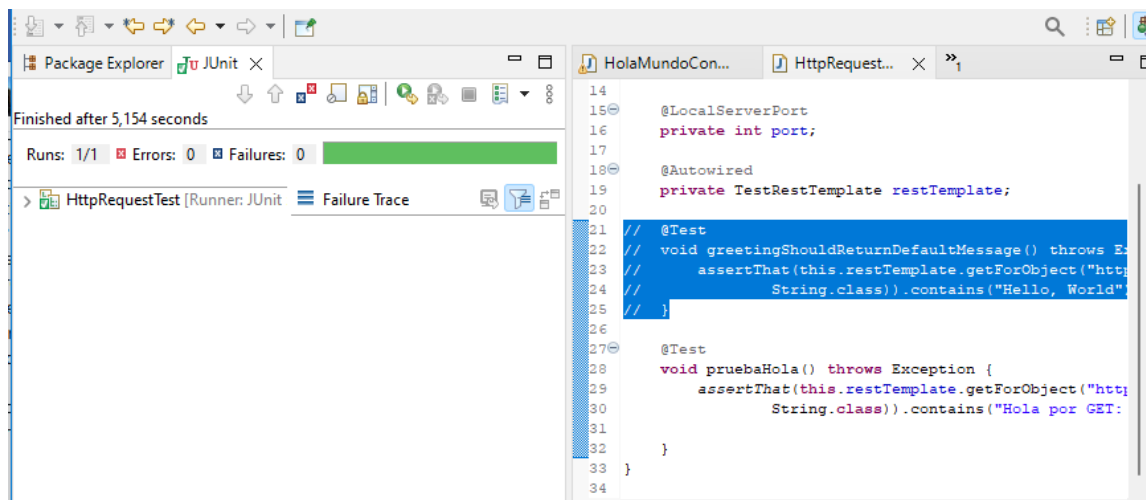
Vemos los resultados en Gradle executions, vemos que hay error, pero tenemos que desplegar demasiadas líneas para ver dónde está el error, aún así tampoco dice el error exacto:

Operation	Duration
Run build	27,192 s
> Calculate build tree task graph	0,875 s
> Load build	3,033 s
Build phase: Run main tasks	16,498 s
Build started for file system watching	0,044 s
Finish root build tree	0,024 s
> Configure build	5,254 s
Run main tasks	16,498 s
Run tasks	16,479 s
Resolve mutations for task :processResources	0,001 s
:test	9,767 s
Snapshot outputs after executing task ':test'	0,106 s
Resolve files of :testRuntimeClasspath	0,001 s
Executing task ':test'	9,491 s
Execute executeTests for :test	9,489 s
Delete old HTML results	0,015 s
com.example.demo.HttpRequestTest	6,964 s
18:44:42.332 [Test worker] INFO org.springframework	0,000 s
18:44:42.625 [Test worker] INFO org.springframework	0,000 s
.	0,000 s
.	0,000 s
^V _' _ _ _ _ _ _ _ _ _ _	0,000 s
((\ _ _ _ _ V _ _ _ _	0,000 s
\ V _ _ _ _ _ _ _) _ _ _	0,000 s
' _ _ _ _ _ _ _ _ _ _ _	0,000 s
===== _ ===== _ /	0,000 s
:: Spring Boot :: (v3.2.0)	0,000 s
.	0,000 s
2023-12-18T18:44:43.547+01:00 INFO 11476	0,000 s
2023-12-18T18:44:43.552+01:00 INFO 11476	0,000 s
2023-12-18T18:44:45.670+01:00 INFO 11476	0,000 s
2023-12-18T18:44:45.687+01:00 INFO 11476	0,000 s
2023-12-18T18:44:45.688+01:00 INFO 11476	0,000 s
2023-12-18T18:44:45.814+01:00 INFO 11476	0,000 s
2023-12-18T18:44:45.817+01:00 INFO 11476	0,000 s
2023-12-18T18:44:46.579+01:00 INFO 11476	0,000 s
2023-12-18T18:44:46.598+01:00 INFO 11476	0,000 s
pruebaHola()	2,289 s
2023-12-18T18:44:48.639+01:00 INFO 11	0,000 s
2023-12-18T18:44:48.640+01:00 INFO 11	0,000 s
2023-12-18T18:44:48.642+01:00 INFO 11	0,000 s
Delete old JUnit XML results	0,001 s
Generate HTML test report for Package com.e	0,021 s
Resolve files of :testRuntimeClasspath	0,001 s

Pruebo ahora con Junit (antes vuelvo a modificar HolMundoController y quito el error):

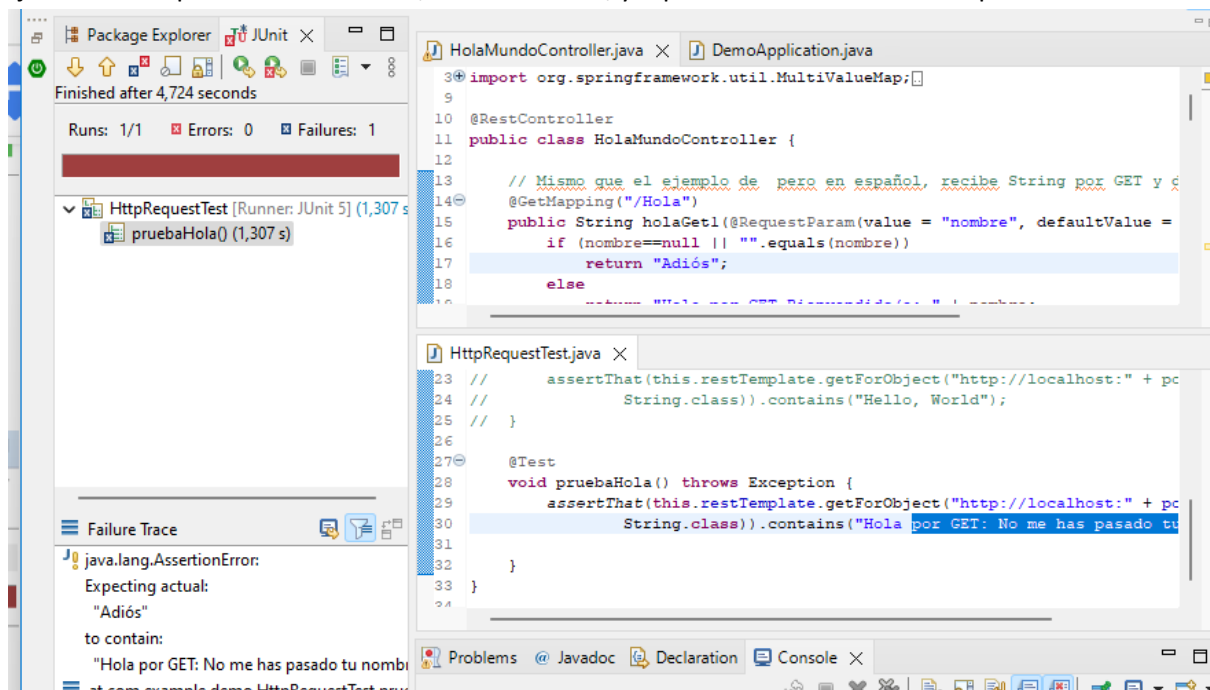


He comentado las líneas que salían en la copia del código, y no sale ningún error, el código es correcto:

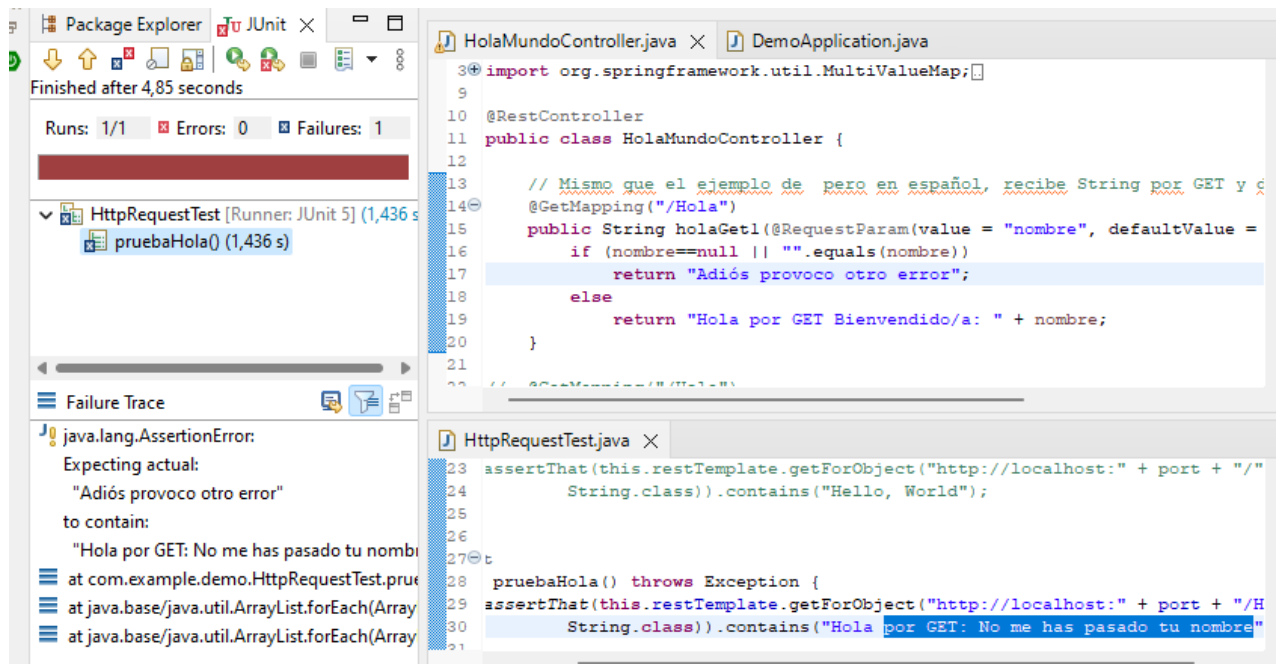


Procedo a realizar pruebas:

- En la clase HolaMundoController, en el getMappint /Hola, en el primer return ponemos “Adiós” y ejecutamos la prueba nuevamente, vemos el error, ya que no devuelve el valor esperado:



Hacemos otras pruebas:



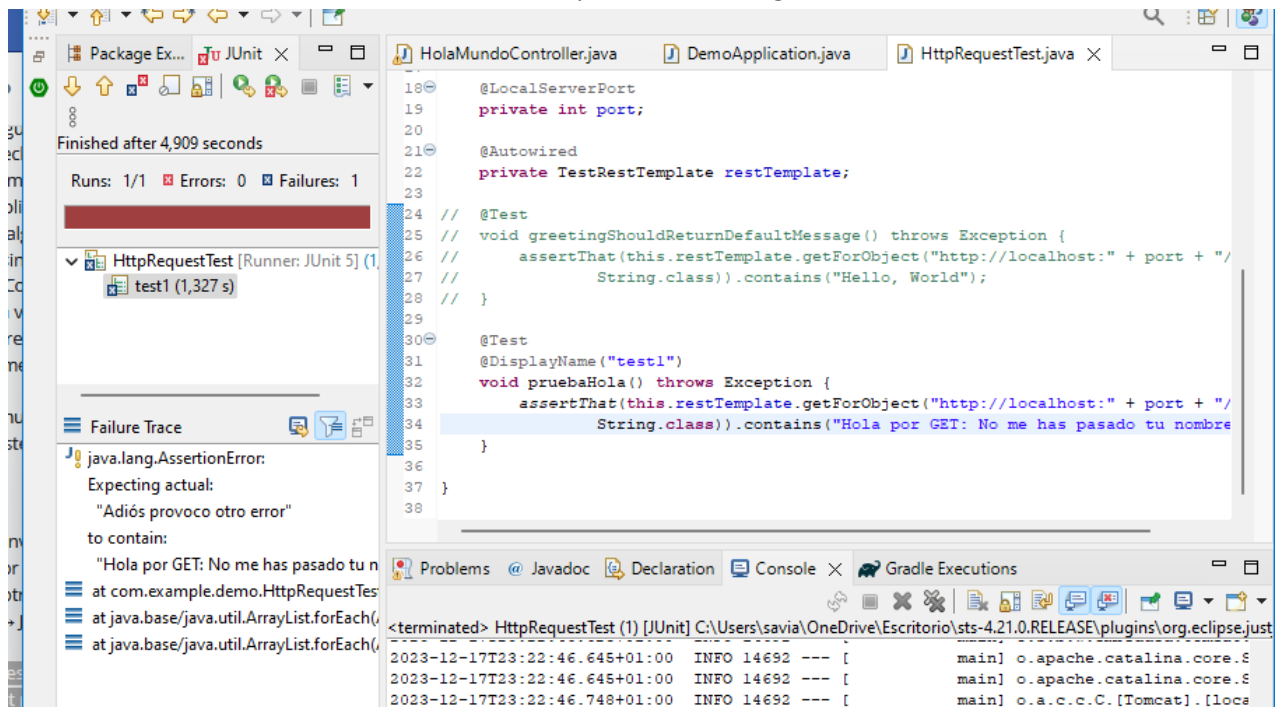
Mucho mejor hacer las pruebas con JUnit, mucho más claro.

Puedes añadir la anotación `@Nested` antes de `@SpringBootTest` y la etiqueta `@DisplayName("Nombre del test")` después de `@Test` para mejorar la salida del Junit test.



Y ejecuto Junit Test nuevamente.

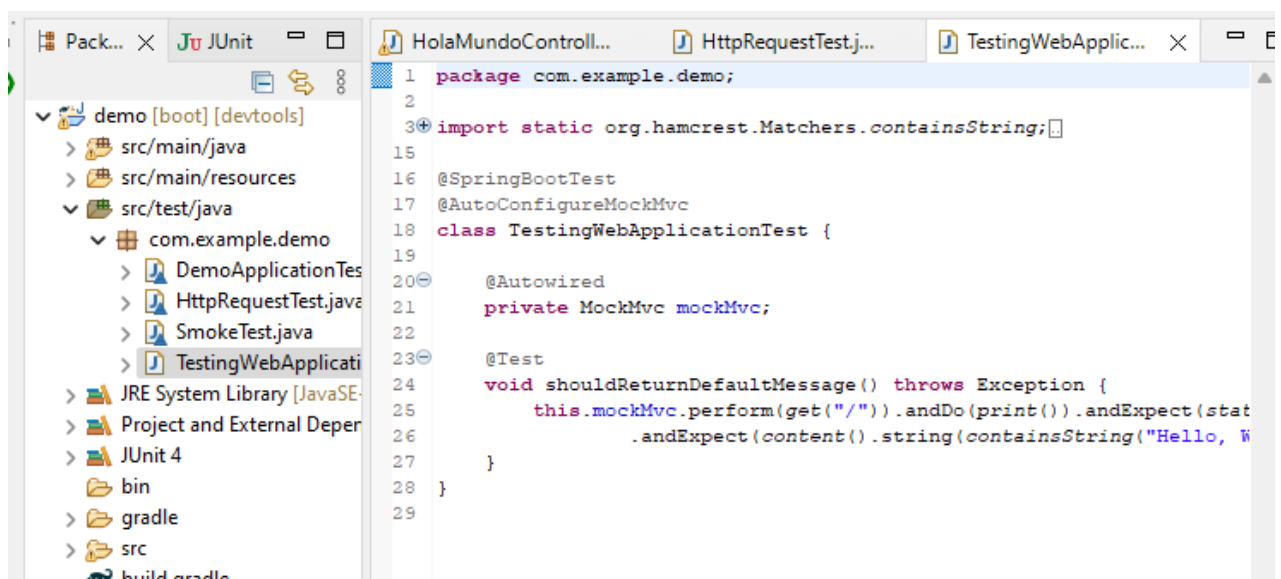
Podemos ver ahora el nombre del test, será útil para cuando tengamos varios test:



Estas dos anotaciones son características que ofrecen funcionalidades específicas para estructurar y personalizar la visualización de tus pruebas.

- `@Nested` se utiliza para crear clases internas anidadas dentro de una clase de prueba. Esto permite organizar y agrupar pruebas relacionadas de una manera más clara y semántica.
- `@DisplayName` se utiliza para personalizar el nombre que se mostrará para una clase de prueba o un método de prueba en los informes y en la salida de la consola.

6. Ahora `TestingWebApplicationTest` pero esta vez sí copia también los imports. Llamará a la función sin levantar el servicio web, como explica el tutorial. El profesor explicará en clase las diferencias con el anterior. Para que sea aún más ligero puedes sustituir `@SpringBootTest` por `@WebMvcTest(HolaMundoController.class)` como sugiere el tutorial.



Cambio la anotación `@SpringBootTest` por `@WebMvcTest(HolaMundoController.class)` :

```

30 import static org.hamcrest.Matchers.containsString;
16
17 // @SpringBootTest
18 @WebMvcTest(HolaMundoController.class)
19 @AutoConfigureMockMvc
20 class TestingWebApplicationTest {
21
22     @Autowired

```

En esta prueba, Spring Boot instancia sólo la capa web en lugar de todo el contexto.

7. Vamos a volver a `HttpRequestTest` para probar algunos más de los puntos de acceso de nuestra aplicación, en concreto el POST que devuelve `String` sin parámetros y con un parámetro. Abajo tienes los ejemplos de los métodos. Sería ya muy avanzado para este momento hacerlo también con JSON, volveremos a ellos en el tema 6.

Hago prueba de `@PostMapping /Hola` sin parámetros. Añado punto de acceso en `HttpRequestTest`:

```

HolaMundoController.java  *HttpRequestTest.java  TestingWebApplicationTetest.java
25 // void greetingShouldReturnDefaultMessage() throws Exception {
26 //     assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/",
27 //         String.class)).contains("Hello, World");
28 // }
29
30 @Test
31 @DisplayName("test1")
32 void pruebaHola() throws Exception {
33     assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/Hola",
34         String.class)).contains("Hola por GET: No me has pasado tu nombre");
35 }
36
37 @Test
38 @DisplayName("Prueba HTTP POST sin param")
39 void saludoPostShouldReturnDefaultMessage() throws Exception {
40     assertThat(this.restTemplate.postForObject("http://localhost:" + port + "/Hola",
41         null, String.class)).contains("Hola por POST. No me has pasado tu nombre");
42 }
43
44 }
45

```

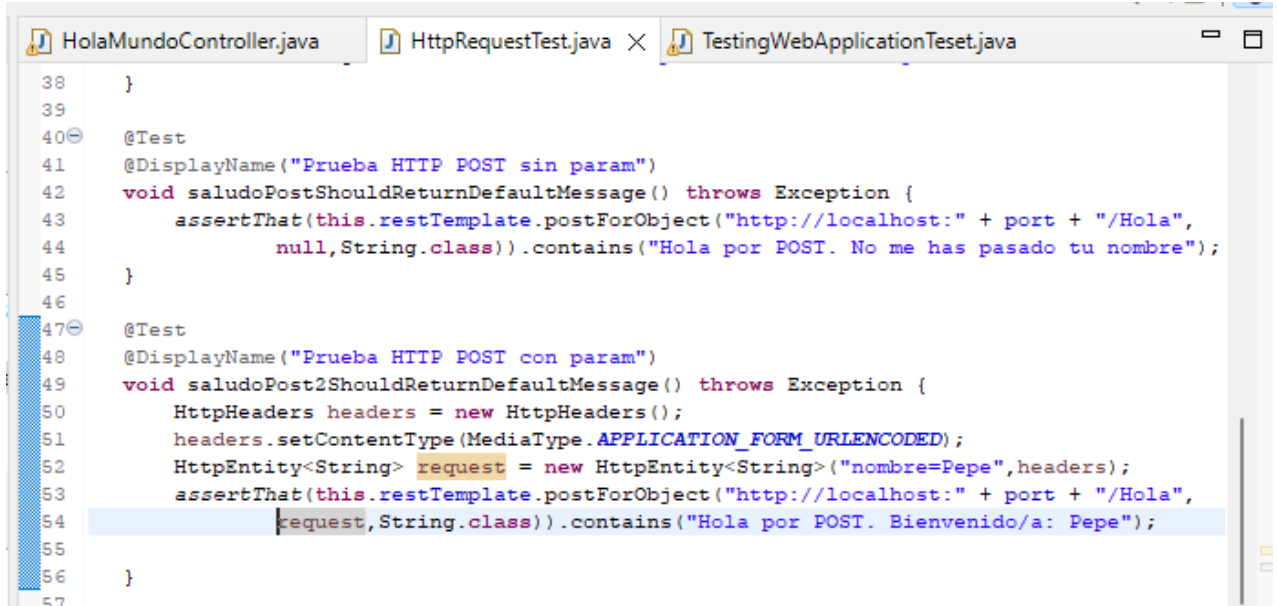
Y creo error en `HolaMundoController` y ejecuto Junit test:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with `HttpRequestTest` selected.
- JUnit Runner:** Shows the test results for `test1 (1,834 s)` and `Prueba HTTP POST sin param (0,067 s)`. The test `Prueba HTTP POST sin param` failed.
- Failure Trace:** Shows the error message: `java.lang.AssertionError: Expecting actual: "Hola por POST soy un fantasma." to contain: "Hola por POST. No me has pasado tu nombre"`. The stack trace includes `at com.example.demo.HttpRequestTest.saludoPost`.
- Code Editor:** Shows the `HolaMundoController` code. The `holoPost1` method is highlighted, which returns `"Hola por POST soy un fantasma."` when `paramMap` is null, and `"Hola por POST. Bienvenido/a: " + paramMap.getFirst("nombre")` otherwise.

Vemos el error en el punto de acceso Prueba HTTP POST sin param, y también observamos que el test del punto de acceso anterior, test1 no tiene errores, ya que lo corregimos.

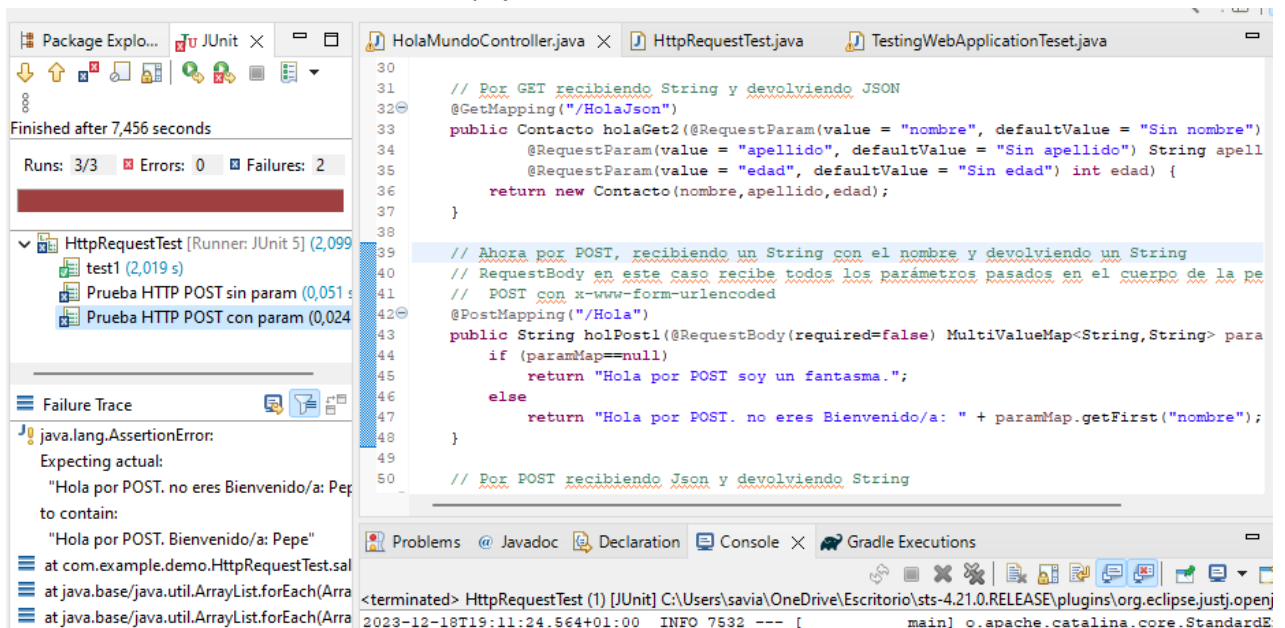
Ahora genero otro punto de acceso en HttpRequestTest para el método @PostMapping /Hola pero esta vez con parámetros:



```

38     }
39
40     @Test
41     @DisplayName("Prueba HTTP POST sin param")
42     void saludoPostShouldReturnDefaultMessage() throws Exception {
43         assertThat(this.restTemplate.postForObject("http://localhost:" + port + "/Hola",
44             null, String.class)).contains("Hola por POST. No me has pasado tu nombre");
45     }
46
47     @Test
48     @DisplayName("Prueba HTTP POST con param")
49     void saludoPost2ShouldReturnDefaultMessage() throws Exception {
50         HttpHeaders headers = new HttpHeaders();
51         headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
52         HttpEntity<String> request = new HttpEntity<String>("nombre=Pepe", headers);
53         assertThat(this.restTemplate.postForObject("http://localhost:" + port + "/Hola",
54             request, String.class)).contains("Hola por POST. Bienvenido/a: Pepe");
55     }
56     }
57
  
```

Genero error en HolaMundoController y ejecuto JUnit test:



```

30
31 // Por GET recibiendo String y devolviendo JSON
32 @GetMapping("/HolaJson")
33 public Contacto holaGet2(@RequestParam(value = "nombre", defaultValue = "Sin nombre")
34     @RequestParam(value = "apellido", defaultValue = "Sin apellido") String apell
35     @RequestParam(value = "edad", defaultValue = "Sin edad") int edad) {
36     return new Contacto(nombre, apellido, edad);
37 }
38
39 // Ahora por POST, recibiendo un String con el nombre y devolviendo un String
40 // RequestBody en este caso recibe todos los parámetros pasados en el cuerpo de la pe
41 // POST con x-www-form-urlencoded
42 @PostMapping("/Hola")
43 public String holaPost1(@RequestBody(required=false) MultiValueMap<String, String> para
44     if (paramMap==null)
45         return "Hola por POST soy un fantasma.";
46     else
47         return "Hola por POST. no eres Bienvenido/a: " + paramMap.getFirst("nombre");
48 }
49
50 // Por POST recibiendo Json y devolviendo String
  
```

Failure Trace

```

java.lang.AssertionError:
Expecting actual:
"Hola por POST. no eres Bienvenido/a: Pepe"
to contain:
"Hola por POST. Bienvenido/a: Pepe"
at com.example.demo.HttpRequestTest.sal
at java.base/java.util.ArrayList.forEach(Arra
at java.base/java.util.ArrayList.forEach(Arra
  
```

terminated> HttpRequestTest (1) [JUnit] C:\Users\savia\OneDrive\Escritorio\sts-4.21.0.RELEASE\plugins\org.eclipse.justj.openj
2023-12-18T19:11:24.564+01:00 INFO 7532 --- [main] o.apache.catalina.core.StandardE

Vemos el error y su localización, también el error anterior ya que no lo hemos corregido.

8. Avanzado: a partir de este punto nos plantea simular que nuestro servicio tiene varias dependencias para ver las pruebas de integración:

- **Copia GreetingController y GreetingService como HolaMundoController y HolaMundoService adaptándolos a nuestro programa. Puedes cambiar el método greet por saluda, por ejemplo. Copio el código nuevo a mi clase HolaMundoController y creo clase nueva HolaMundoService:**

The screenshot shows two Java files in an IDE. The left file, `HolaMundoController.java`, contains the following code:

```
1 package com.example.demo;
2
3 import org.springframework.util.MultiValueMap;
4
5
6
7
8
9
10
11
12 @RestController
13 public class HolaMundoController {
14
15     private final HolaMundoService service;
16
17     public HolaMundoController(HolaMundoService service) {
18         this.service = service;
19     }
20
21     @RequestMapping("/saludando")
22     public @ResponseBody String saludando() {
23         return service.saludo();
24     }
25
26     // Mismo que el ejemplo de pero en español, recibe String
27     @GetMapping("/Hola")
28     public String holaGet1(@RequestParam(value = "nombre", def
```

The right file, `HolaMundoService.java`, contains the following code:

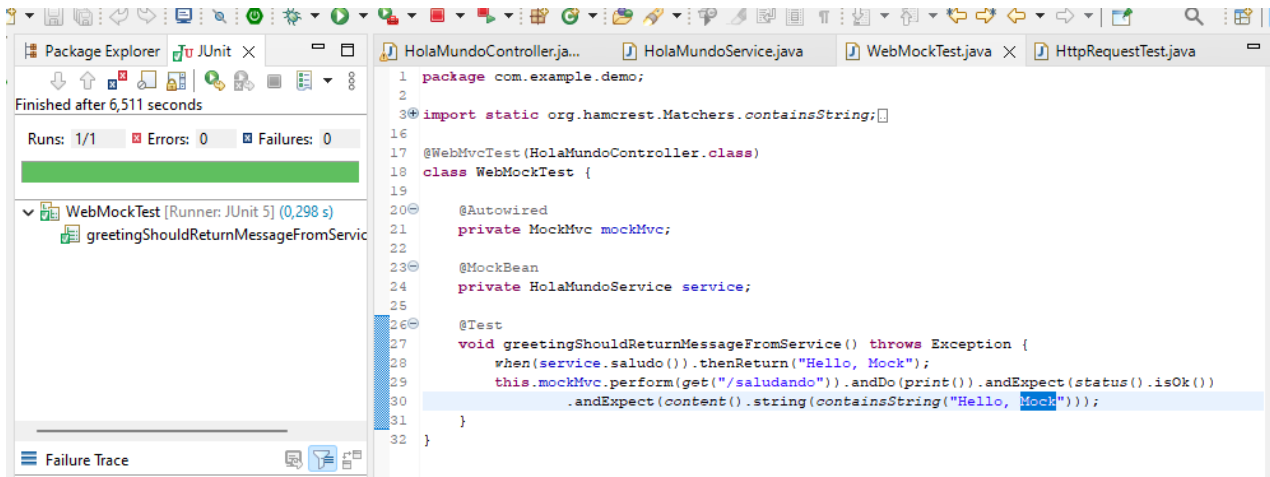
```
1 package com.example.demo;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class HolaMundoService {
7     public String saludo() {
8         return "Hola, Mundo";
9     }
10 }
```

- **Copia WebMockTest tal cual y adapítala a nuestros nombres, método saluda y texto que devuelve. Pruébalo, no desesperes si no consigues que de válido y documenta lo realizado. En clase el profesor explicará qué quiere decir cada parte.**
Y ahora en la carpeta de test, creo esta clase, cambiando los nombres del ejemplo a los de mi código:

The screenshot shows the `WebMockTest.java` file in an IDE. The code is as follows:

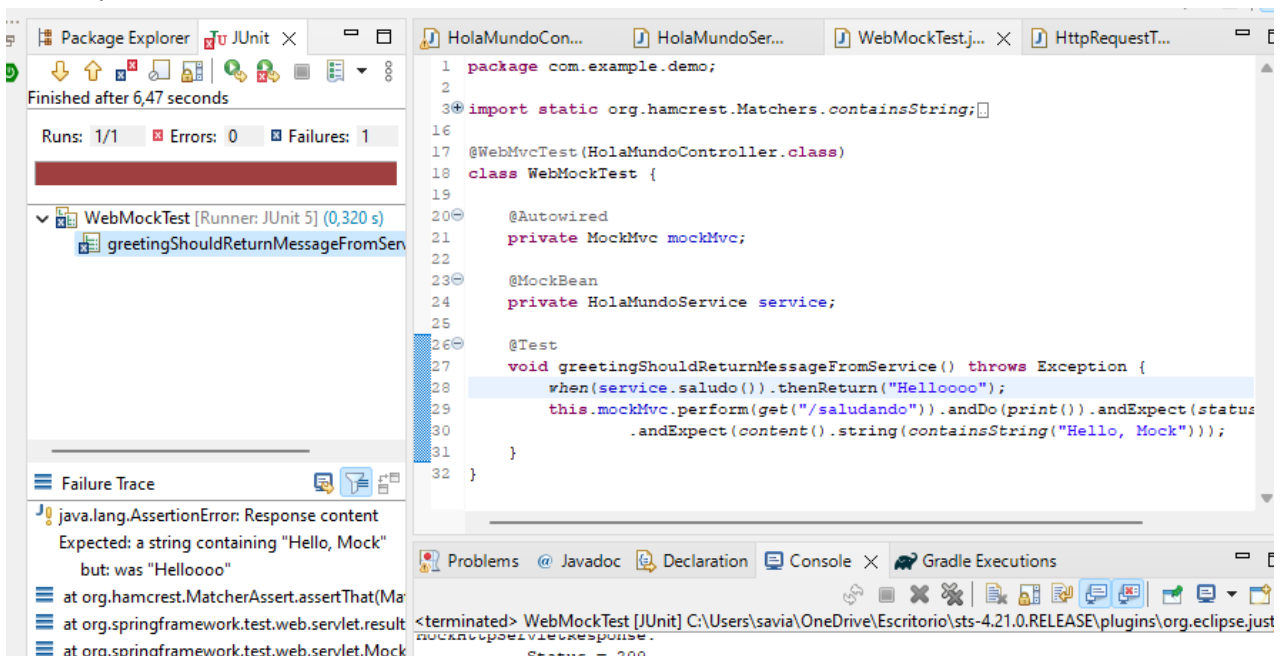
```
1 package com.example.demo;
2
3 import static org.hamcrest.Matchers.containsString;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 @WebMvcTest(HolaMundoController.class)
18 class WebMockTest {
19
20     @Autowired
21     private MockMvc mockMvc;
22
23     @MockBean
24     private HolaMundoService service;
25
26     @Test
27     void greetingShouldReturnMessageFromService() throws Exception {
28         when(service.saludo()).thenReturn("Hello, Mock");
29         this.mockMvc.perform(get("/saludando")).andDo(print()).andExpect(status().isOk())
30             .andExpect(content().string(containsString("Hello, Mock")));
31     }
32 }
```

Y ejecuto Junit test y todo correcto:



Voy a intentar provocar un error, aunque no entiendo bien el código. Cambio el código en la clase WebMockTest, poniendo que recibe Heloooo pero que espera Hello, Mock.

No entiendo bien porque se hace en esta misma clase la prueba cuando en las pruebas anteriores provocaba los errores en la clase del controlador:



AMPLIACIÓN

Voy a conectar mi STS a mi GitHub para así tener el STS de casa sincronizado con el de clase.

Primero, desde GitHub creo un repositorio (correspondiente a un proyecto de STS):

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

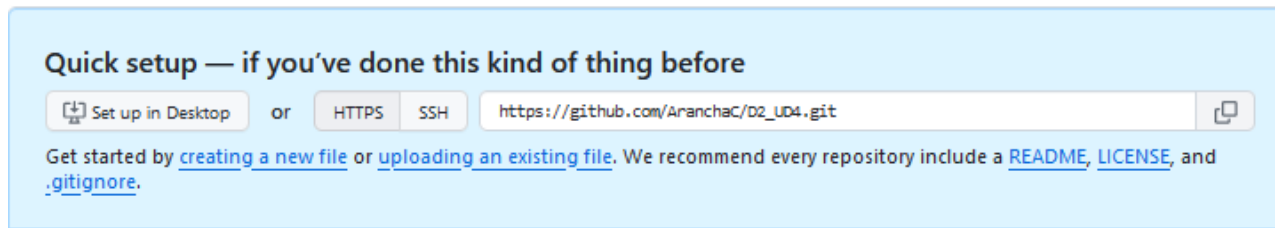
Required fields are marked with an asterisk (*).

Owner * AranchaC / Repository name * p2_UD4
 ✓ D2_UD4 is available.

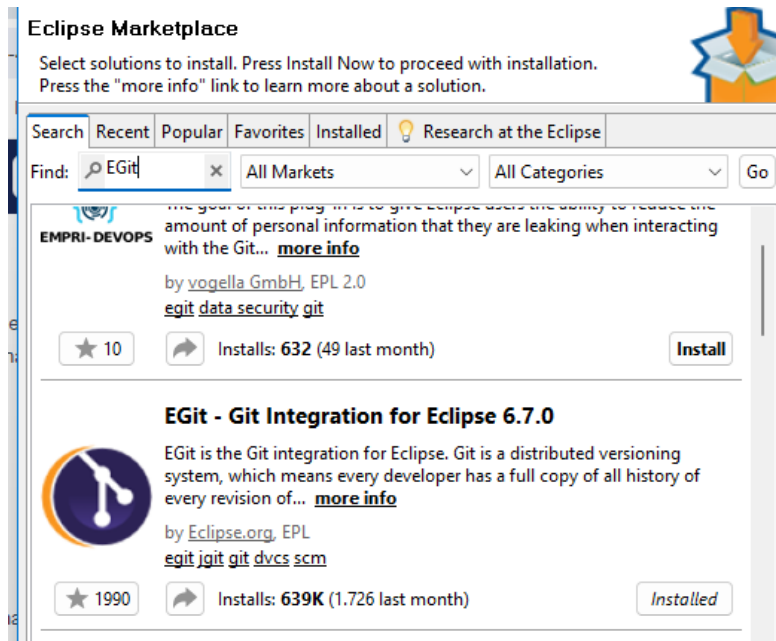
Great repository names are short and memorable. Need inspiration? How about [literate-giggle](#) ?

Description (optional)

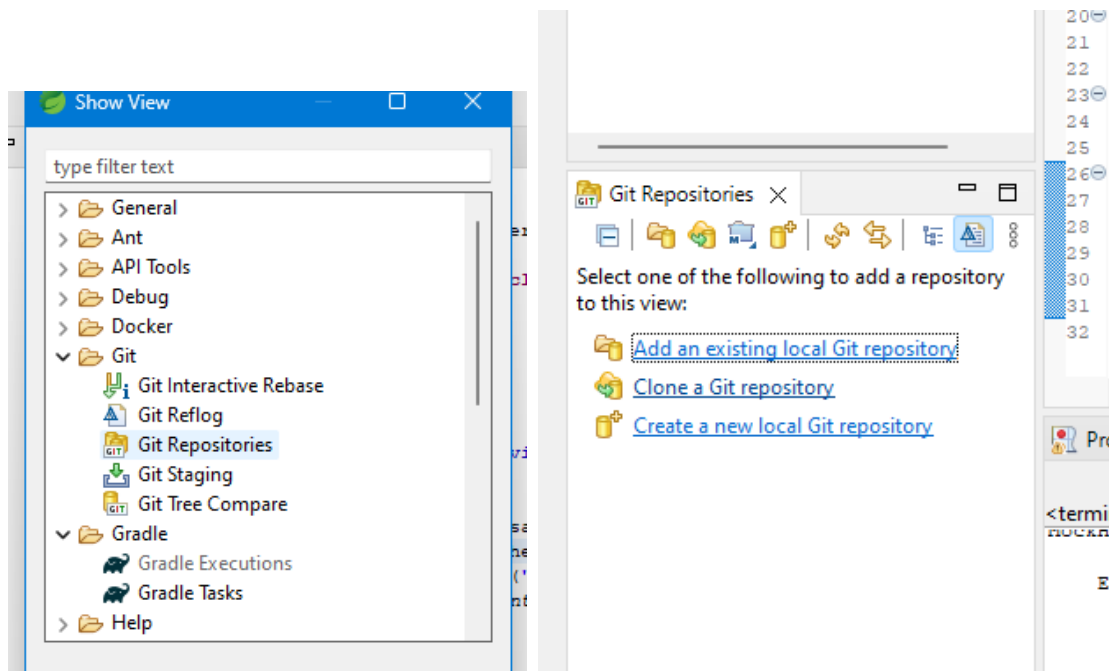
Copio la url del repositorio:

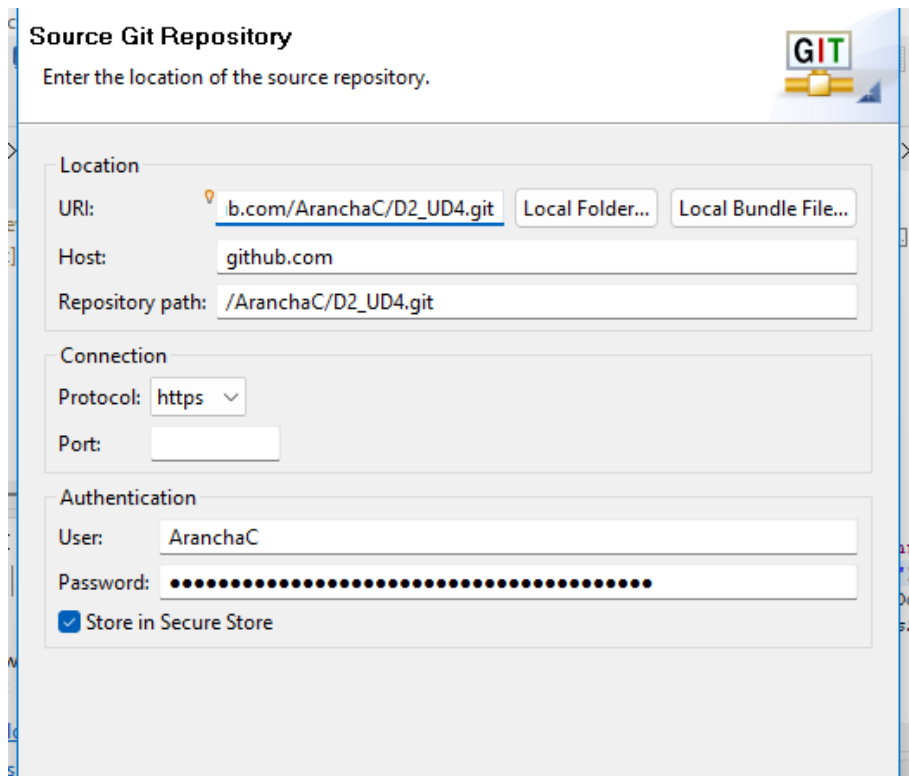


Y ahora desde STS, vemos desde marketplace si tenemos el plugin EGit instalado y si no lo instalamos:

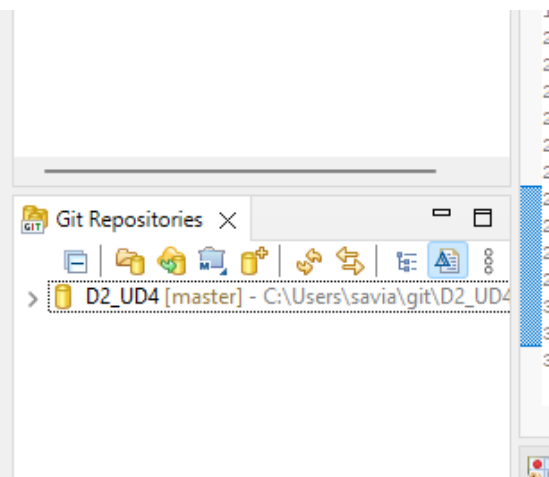
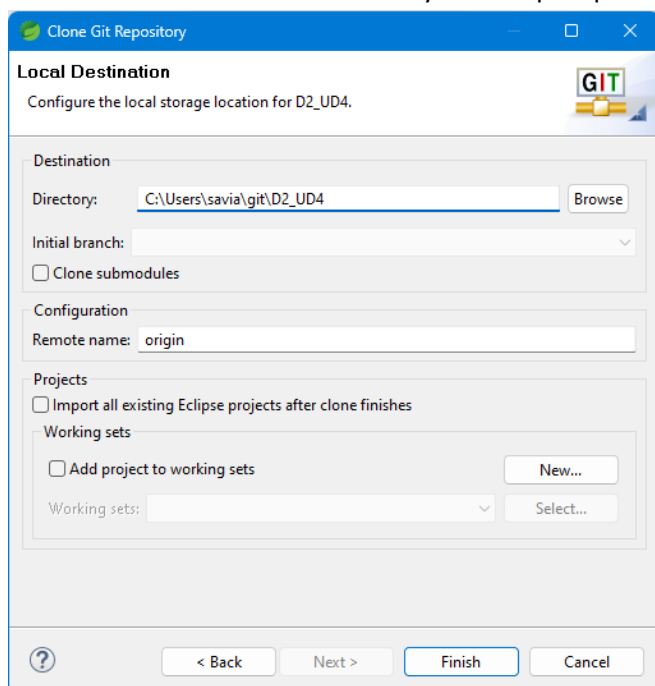


Nos vamos a la perspectiva Git, y clonamos el repositorio que hemos creado y copiado, poniendo nuestros datos de usuario de GitHub:

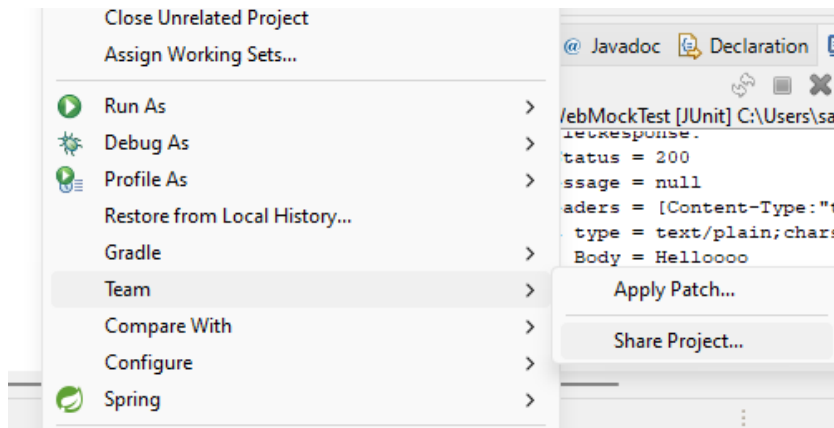




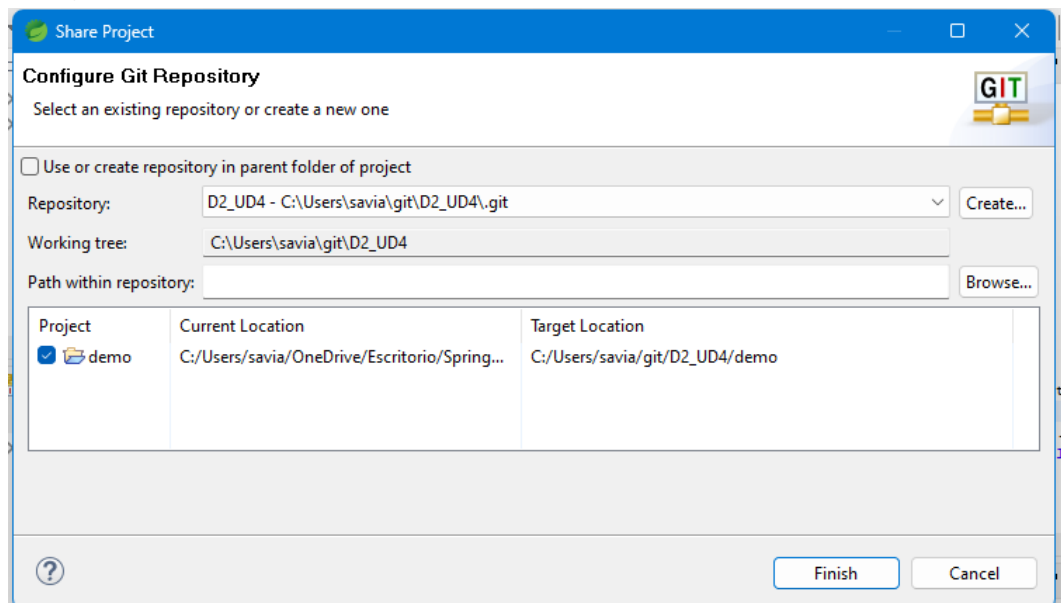
Y damos a next hasta darle a finish y vemos que aparece nuestro repositorio:



Ahora conectamos nuestro proyecto a este repositorio. Click derecho en el proyecto, team y share Project:



Seleccionamos nuestro repositorio y damos a finish (cuando tengamos varios repositorios, aparecerán todos):



Y ahora se ha conectado el proyecto al repositorio, pero tenemos que subir cambios.

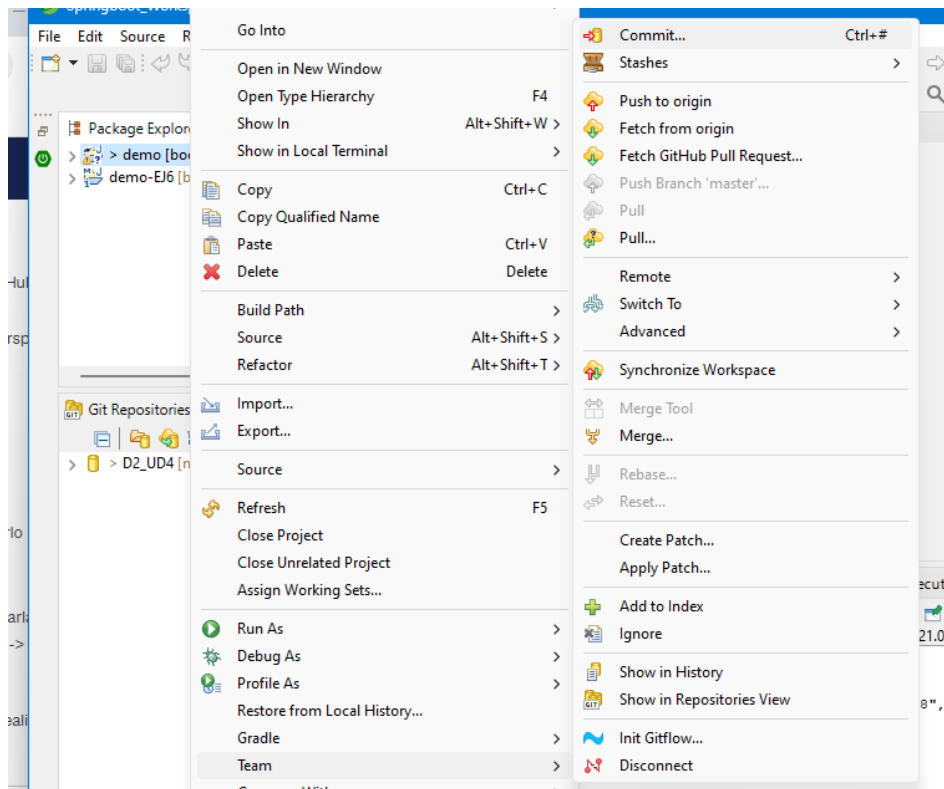
Los pasos siempre serán:

- git pull: descargar cambios (cuando vayamos haciendo más cosas).

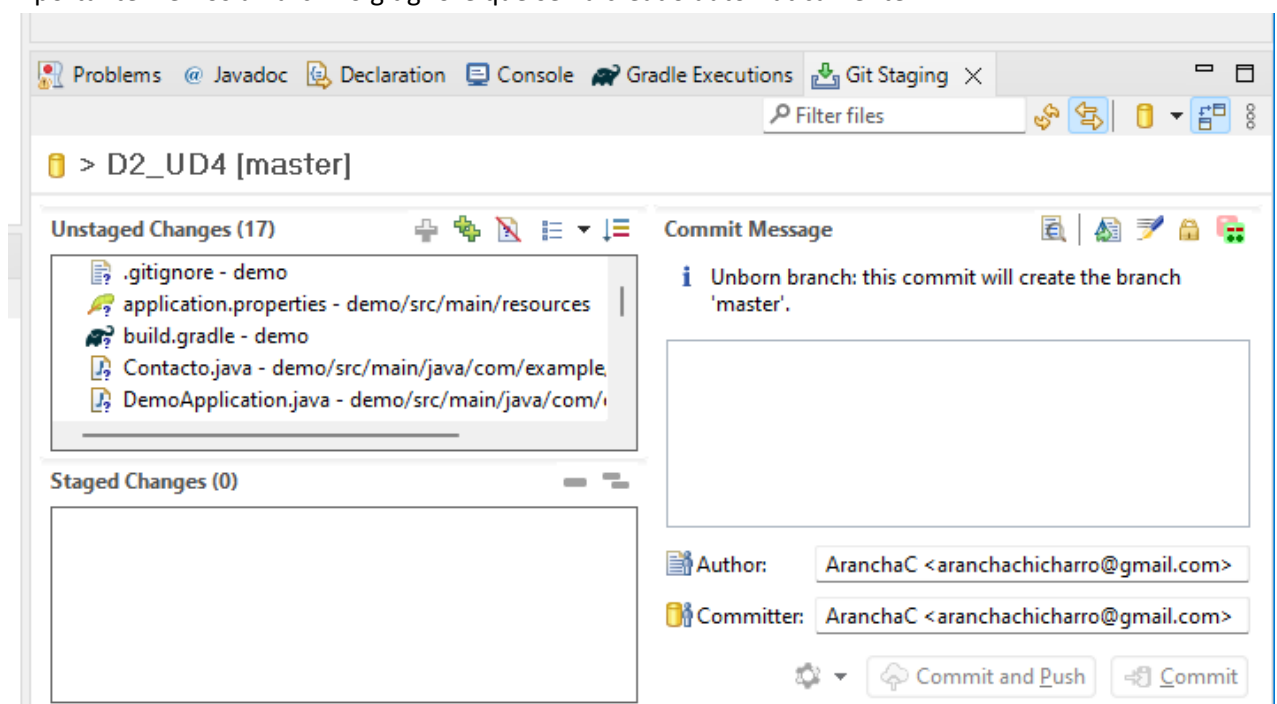
Muy importante hacer lo primero, pues si hacemos cambios sin habernos descargado antes otros existentes se provoca un descuadre importante. Por ejemplo, yo ahora como tendré los sts de casa y clase sincronizados, cada vez que abra cada sts tengo que hacer git pull para descargar los cambios que haya hecho en el otro sts y luego poder seguir trabajando.

- commit: preparar cambios:
- git push: subir cambios.

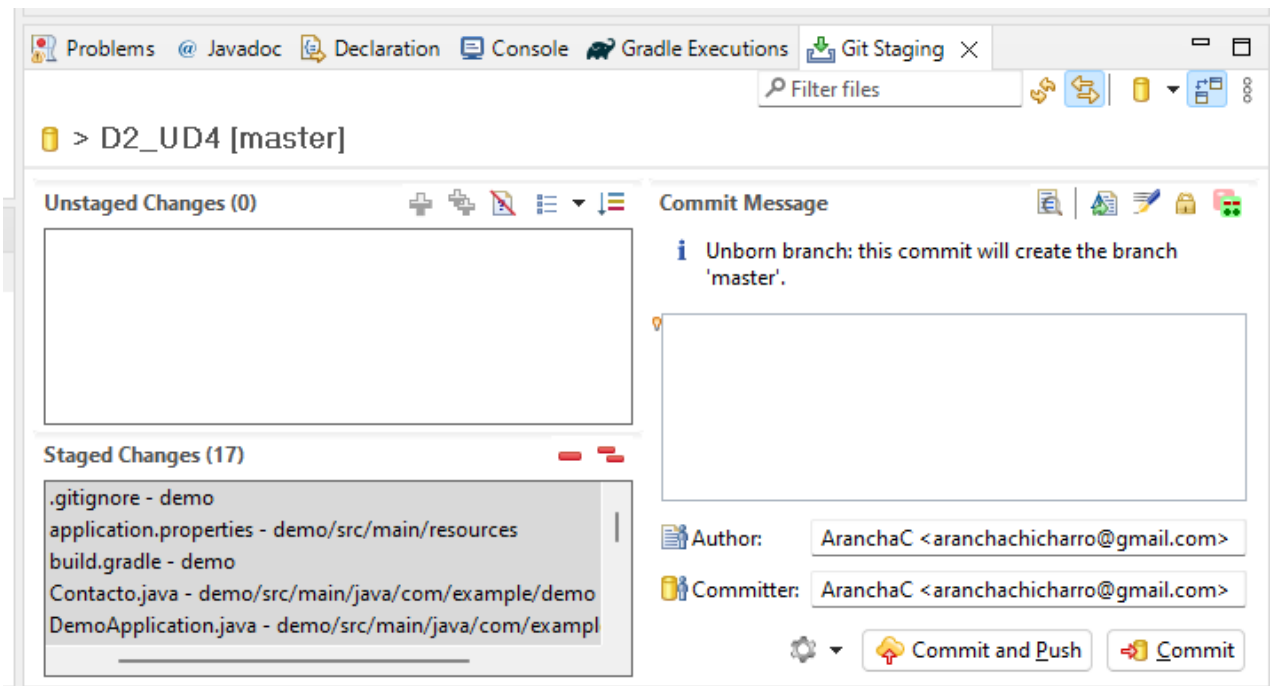
Entonces, desde Team, ahora nos aparecen las funcionalidades de Git, le damos a commit y vemos lo que aparece:



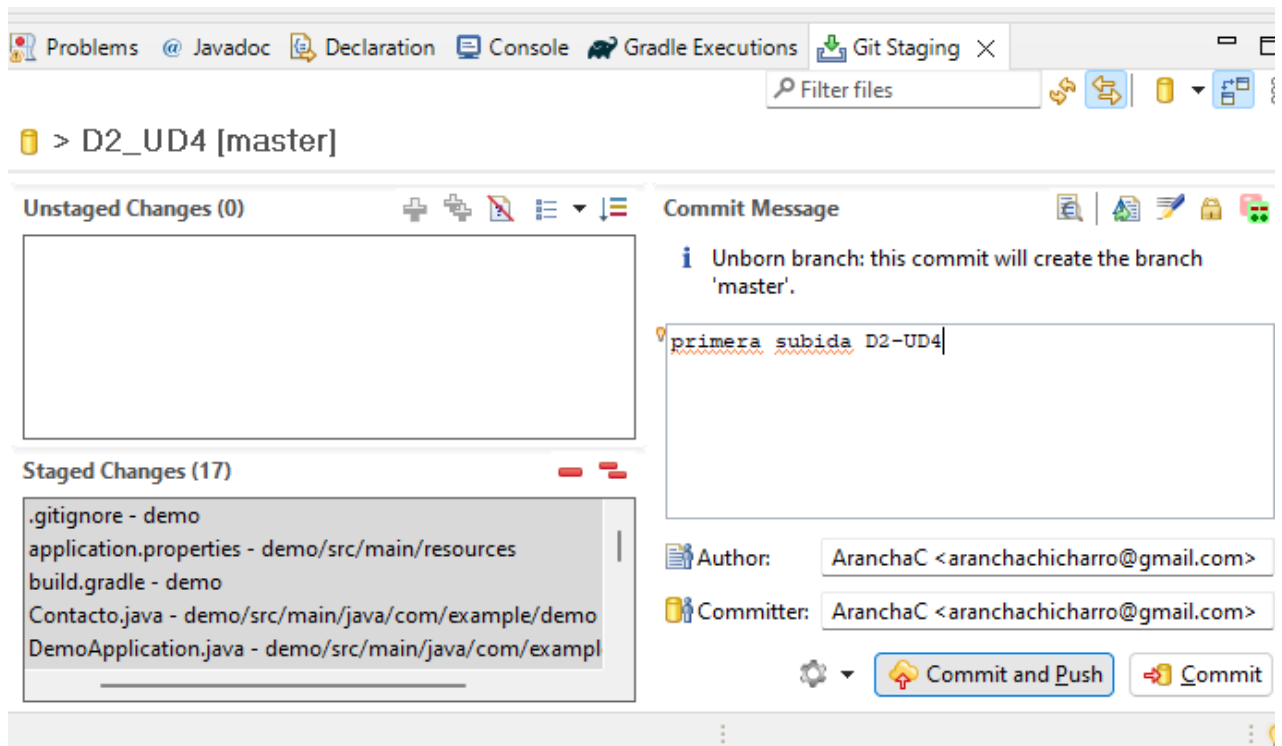
En staged changes, vemos todos los cambios que hay que preparar para subir al repositorio.
importante: vemos un archivo gitignore que se ha creado automáticamente:



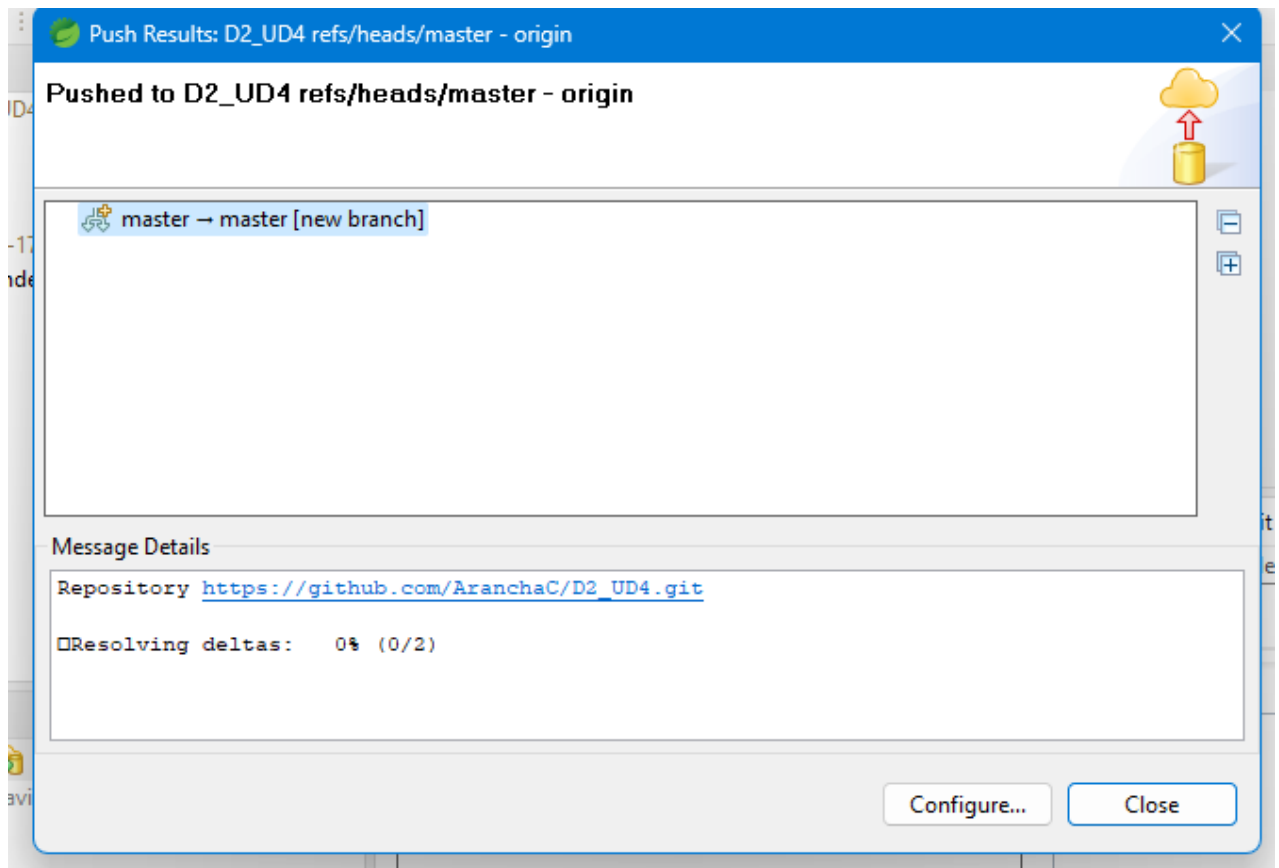
Le damos al icono de los dos “+” verdes y los cambios pasan al cuadro de texto inferior:



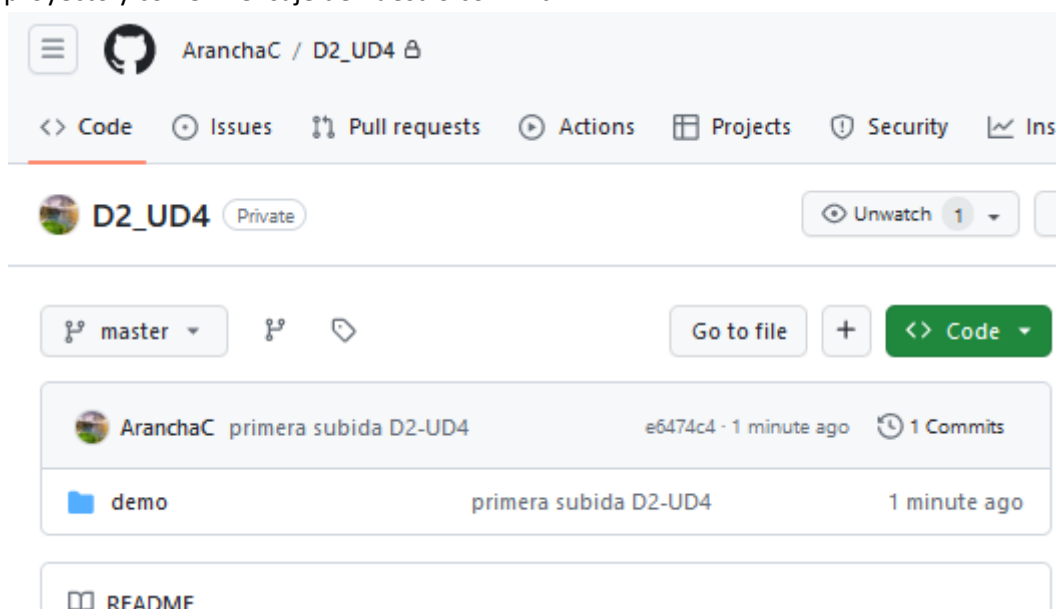
Y ahora en el campo de texto de la derecha, ponemos un mensaje a modo de información que será la etiqueta de nuestro cambio, que se verá reflejado así en GitHub:



Y por último damos a commit and push y si todo sale bien, nos aparece esta ventana a modo de confirmación:



Ahora vamos a nuestra página de Git Hub y comprobamos que nuestro repositorio tiene nuestro proyecto y con el mensaje de nuestro commit:



Accedemos y vemos el archivo gitignore por curiosidad:

The screenshot shows a GitHub repository page for 'D2_UD4 / demo'. The repository is owned by 'AranchaC' and has a commit titled 'primera subida D2-UD4'. The file list on the left includes: '..', 'gradle/wrapper', 'src', '.gitignore', 'build.gradle', 'gradlew', 'gradlew.bat', and 'settings.gradle'. The right pane shows the content of the '.gitignore' file, which is 37 lines long (33 loc) and 444 bytes. The file contains various ignore patterns for a Gradle project.

master D2_UD4 / demo / .gitignore

AranchaC primera subida D2-UD4

Code Blame 37 lines (33 loc) • 444 Bytes

```
1  HELP.md
2  .gradle
3  build/
4  !gradle/wrapper/gradle-wrapper.jar
5  !**/src/main/**/build/
6  !**/src/test/**/build/
7
8  ### STS ###
9  .apt_generated
10 .classpath
11 .factorypath
12 .project
13 .settings
14 .springBeans
15 .sts4-cache
16 bin/
17 !**/src/main/**/bin/
18 !**/src/test/**/bin/
19
20 ### IntelliJ IDEA ###
21 .idea
22 *.iws
23 *.iml
24 *.ipr
25 out/
26 !**/src/main/**/out/
27 !**/src/test/**/out/
```