Crear primeros Servicios

Utilizando : Spring WEB

AR

MAVEN

Spring JPA

JAKARTA VALIDATION

PostgreSQL,

Windows 10

SWAGGER

Lombok, Spring Boot

JAVA

MVC

@Anotation´s

JUnit

Jacoco

POSTMAN

Mockito

Spring Tools 4

PARTE 3

M V C

En esta parte del curso realizaremos las siguientes tareas que nos permiten complementar nuestro trabajo de forma profesional, tomando en cuenta las mejores practicas de desarrollo.

1. **Configuración de dependencias para pruebas unitarias (Junit en POM.xml)**
2. **Generación de clases de prueba unitaria de servicios (Solo la clase Controller), los demás casos de prueba de Services y Repositorio se dejan de tarea por si desean practicar.**
3. **Configuración de dependencias para cobertura de código (Jacoco en POM.xml)**
4. **Generación de reporte de cobertura de código probado (de la clase Controller), los demás clases Services y Repositorio se dejan de tarea por si desean practicar.**
5. **Realizar pruebas funcionales usando POSTMAN.**
6. **Generando evidencias de pruebas unitarias y funcionales.**
7. **Validación de deuda técnica sobre la codificación.**
8. **Corrección de deuda técnica sobre codificación**

- **Quiero hacer mención que las pruebas unitarias y las pruebas funcionales son diferentes, aunque muchos lideres consideran que son lo mismo para un desarrollador, no lo son, y aquí hago la diferencia.**

- **Las pruebas unitarias automáticas usando Junit y Mockito nos muestran una simulación de comportamiento y sirven para generar el reporte de cobertura de código codificado con las excepciones y comportamiento esperado., mientras que las pruebas funcionales unitarias son las que se realizan mediante herramientas diferentes como un POSTMAN, CURL..etc. Y ponen en funcionamiento todos los componentes Controladores, servicios y repositorios. Por lo que las pruebas funcionales pueden desprender acciones de corrección al código.**

Con este entendido, vayamos a continuación con las acciones arriba mencionadas.

Junit, Mockito, Jupiter

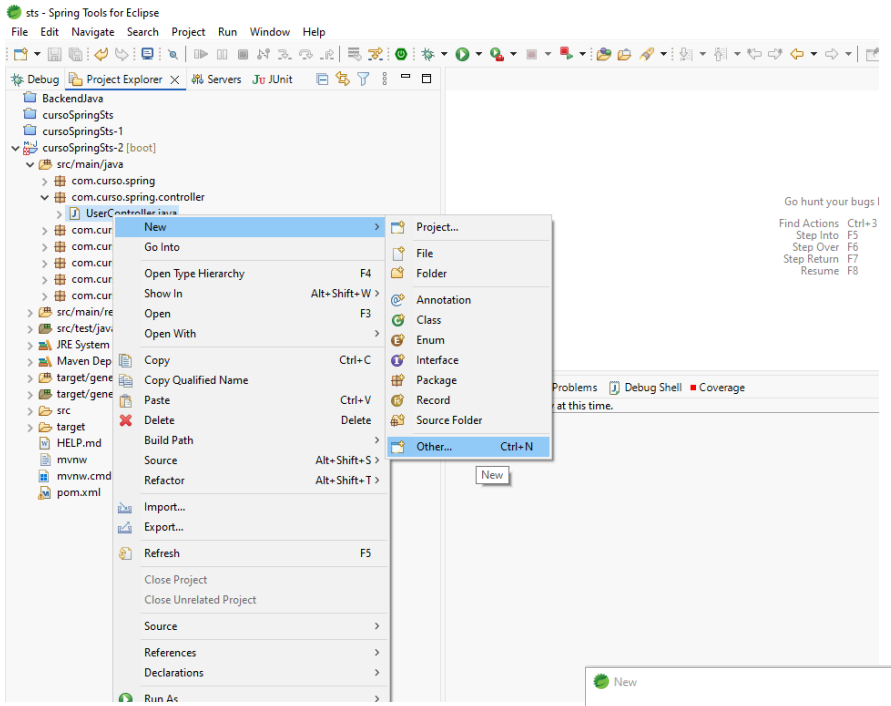
Para la configuración de dependencias de Junit, Mockito y Jupiter se validará que se tenga registradas las librerías en el POM .

```
<!-- Spring Boot Test -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
<dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.12.2</version>
        <scope>test</scope>
</dependency>
<!-- Mockito -->
<dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>5.17.0</version>
        <scope>test</scope>
</dependency>
```

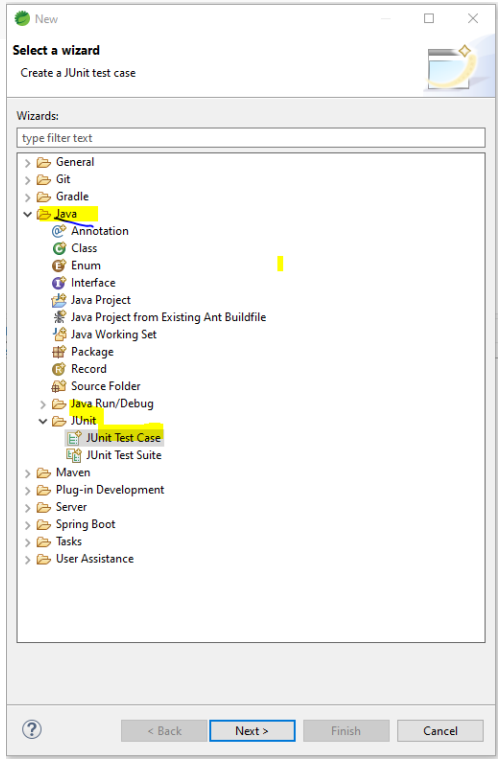
Crear nuestra clase de pruebas

UserControllerTest.java

Seleccionamos la clase UserController y hacemos click derecho sobre ella.
Seleccionamos New – Other y se abre una nueva ventana.



Seleccionamos Java – Junit – Junit Test Case y damos click en Next



Definir clase a Inyectar y Clase que se realiza el Mock

Insertamos el siguiente código

```
@ExtendWith(MockitoExtension.class)
class UserControllerTestBk {

@InjectMocks
private UserController userController;


@Mock
private ServiceUser serviceUser;
```

Definir método de prueba Exitoso

Y comenzamos con nuestro casos de prueba Create

- **Se declara usuario a dar de alta**
- **Se declara retornar el usuario cuando se invoque el servicio** `userController.createUser(user)`
- **Se obtiene el valor del servicio servicio** `userController.createUser(user)`
- **Se arman las variables de retono.**
- **Se realizan las validaciones sobre las respuesta generadas.**

```
@Test
void createUser_Returns201_WhenUserIsSavedSuccessfully() throws Exception {
// Arrange
User user = new User(1, "Juan", "juan@example.com", 1, "securePass123", 1);
when(serviceUser.save(any(User.class))).thenReturn(user);


// Act
ResponseEntity<ApiResponse> response = userController.createUser(user);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuario creado correctamente", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}
```

Definir método de prueba Exception

- **Se declara usuario a dar de alta**
- **Se declara retornar el usuario cuando se invoque el servicio** `userController.createUser(user)` que en este caso será una excepción.
- **Se obtiene el valor del servicio servicio** `userController.createUser(user)`
- **Se arman las variables de retorno.**
- **Se realizan las validaciones sobre las respuesta generadas.**

Así para cada caso de prueba necesario y requerido se debe realizar cada método.

```
@Test
void createUser_Returns400_WhenUsuarioExistenteExceptionIsThrown() throws
Exception {
// Arrange
User user = new User(1, "Juan", "juan@example.com", 1, "securePass123", 1);
when(serviceUser.save(any(User.class))).thenThrow(new
UsuarioExistenteException("Usuario ya existe"));


// Act
ResponseEntity<ApiResponse> response = userController.createUser(user);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(400, result.get("Estatus"));
assertEquals("Usuario ya existe", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}
```

A continuación la clase completa para realizar las pruebas

1

```java
@ExtendWith(MockitoExtension.class)
class UserControllerTestBk {
@InjectMocks
private UserController userController;


@Mock
private ServiceUser serviceUser;

@Test
void createUser_Returns201_WhenUserIsSavedSuccessfully() throws Exception {
// Arrange
User user = new User(1, "Juan", "juan@example.com", 1, "securePass123", 1);
when(serviceUser.save(any(User.class))).thenReturn(user);


// Act
ResponseEntity<ApiResponse> response = userController.createUser(user);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuario creado correctamente", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}

@Test
void createUser_Returns400_WhenUsuarioExistenteExceptionIsThrown() throws Exception {
// Arrange
User user = new User(1, "Juan", "juan@example.com", 1, "securePass123", 1);
when(serviceUser.save(any(User.class))).thenThrow(new UsuarioExistenteException("Usuario ya
existe"));


// Act
ResponseEntity<ApiResponse> response = userController.createUser(user);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(400, result.get("Estatus"));
assertEquals("Usuario ya existe", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}
//-----------------------------------------------------------
@Test
void getUserById_Returns201_WhenUserExists() {
// Arrange
Integer userId = 1;
User user = new User(userId, "Juan", "juan@example.com", 1, "securePass123", 1);
when(serviceUser.getUserById(userId)).thenReturn(user);


// Act
ResponseEntity<ApiResponse> response = userController.getUserById(userId);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuario obtenido correctamente", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}
```

2

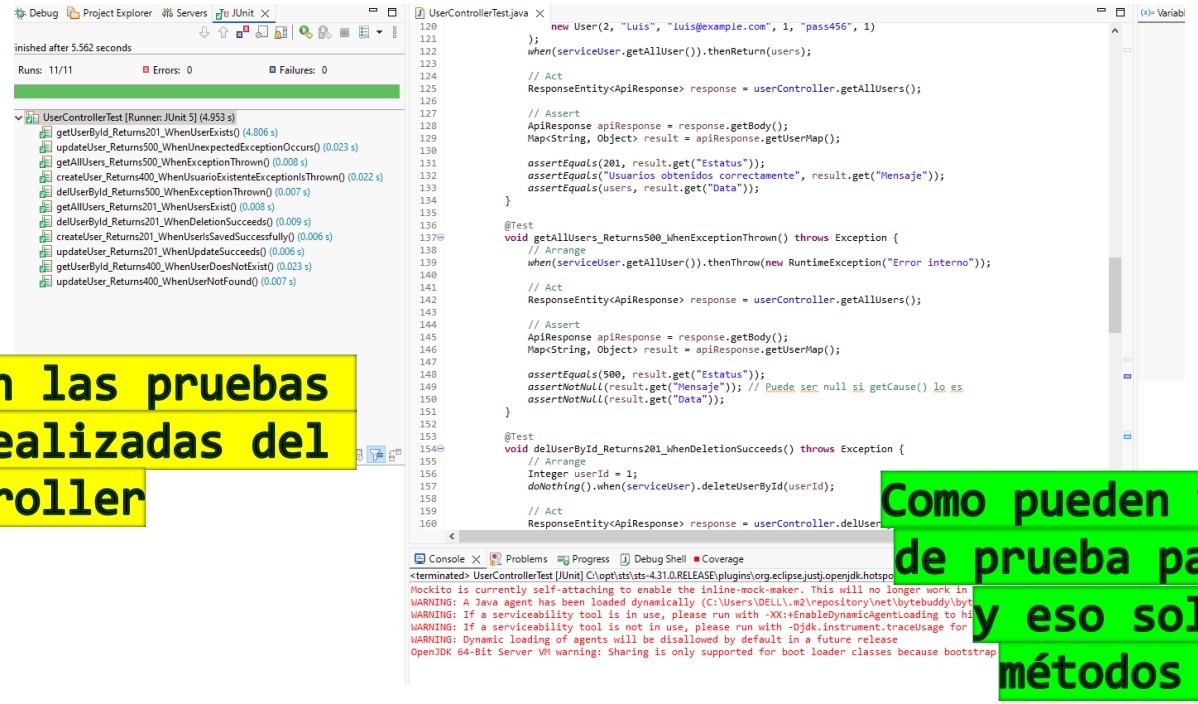
```java
@Test
void getUserById_Returns400_WhenUserDoesNotExist() {
// Arrange
Integer userId = 999;
when(serviceUser.getUserById(userId)).thenThrow(new UsuarioNoExisteteException("Usuario no
encontrado"));


// Act
ResponseEntity<ApiResponse> response = userController.getUserById(userId);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(400, result.get("Estatus"));
assertEquals("Usuario no encontrado", result.get("Mensaje"));
assertNull(result.get("Data"));
}
@Test
void getAllUsers_Returns201_WhenUsersExist() throws Exception {
// Arrange
List<User> users = List.of(
new User(1, "Ana", "ana@example.com", 1, "pass123", 1),
new User(2, "Luis", "luis@example.com", 1, "pass456", 1)
);
when(serviceUser.getAllUser()).thenReturn(users);

// Act
ResponseEntity<ApiResponse> response = userController.getAllUsers();

// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuarios obtenidos correctamente", result.get("Mensaje"));
assertEquals(users, result.get("Data"));
}

@Test
void getAllUsers_Returns500_WhenExceptionThrown() throws Exception {
// Arrange
when(serviceUser.getAllUser()).thenThrow(new RuntimeException("Error interno"));


// Act
ResponseEntity<ApiResponse> response = userController.getAllUsers();

// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(500, result.get("Estatus"));
assertNotNull(result.get("Mensaje")); // Puede ser null si getCause() lo es
assertNotNull(result.get("Data"));
}
@Test
void delUserById_Returns201_WhenDeletionSucceeds() throws Exception {
// Arrange
Integer userId = 1;
doNothing().when(serviceUser).deleteUserById(userId);
// Act
ResponseEntity<ApiResponse> response = userController.delUserById(userId);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuario eliminado correctamente", result.get("Mensaje"));
assertEquals(userId, result.get("Data"));
```

3

```java
@Test
void delUserById_Returns500_WhenExceptionThrown() throws Exception {
// Arrange
Integer userId = 999;
doThrow(new Exception("Error al eliminar")).when(serviceUser).deleteUserById(userId);


// Act
ResponseEntity<ApiResponse> response = userController.delUserById(userId);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(500, result.get("Estatus"));
assertNotNull(result.get("Mensaje")); // Puede ser null si getCause() lo es
assertNull(result.get("Data"));
}
@Test
void updateUser_Returns201_WhenUpdateSucceeds() {
// Arrange
Integer userId = 1;
User user = new User(userId, "Carlos", "carlos@example.com", 1, "securePass", 1);
//doNothing().when(serviceUser).updateUser(userId, user);
// Act
ResponseEntity<ApiResponse> response = userController.updateUser(userId, user);


// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(201, result.get("Estatus"));
assertEquals("Usuario actualizado correctamente", result.get("Mensaje"));
assertEquals(user, result.get("Data"));
}

@Test
void updateUser_Returns400_WhenUserNotFound() {
// Arrange
Integer userId = 999;
User user = new User(userId, "Desconocido", "no@existe.com", 1, "pass", 1);
try {
doThrow(new UsuarioNoExisteteException("Usuario no
existe")).when(serviceUser).updateUser(userId, user);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
// Act
ResponseEntity<ApiResponse> response = userController.updateUser(userId, user);

// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();


assertEquals(400, result.get("Estatus"));
assertEquals("Usuario no existe", result.get("Mensaje"));
assertNull(result.get("Data"));
}
@Test
void updateUser_Returns500_WhenUnexpectedExceptionOccurs() throws Exception {
// Arrange
Integer userId = 2;
User user = new User(userId, "Error", "error@example.com", 1, "fail", 1);
doThrow(new Exception("Falla inesperada")).when(serviceUser).updateUser(userId, user);
// Act
ResponseEntity<ApiResponse> response = userController.updateUser(userId, user);

// Assert
ApiResponse apiResponse = response.getBody();
Map<String, Object> result = apiResponse.getUserMap();

assertEquals(500, result.get("Estatus"));
assertNotNull(result.get("Mensaje")); // Puede ser null si getCause() lo es
assertNull(result.get("Data"));
```

Lanzar pruebas unitarias JUnit

1. **Abrimos la clase** UserControllerTest
2. **Haciendo click derecho sobre el código de la clase, seleccionamos Run as -- Junit Test**



Pantalla con las pruebas unitarias realizadas del controller

Como pueden ver son 11 métodos de prueba para un simple CRUD y eso solo contando los métodos del servicio.

Como les comente anteriormente, los demás métodos de componentes Service y Repository los dejo de tarea para que prueben.

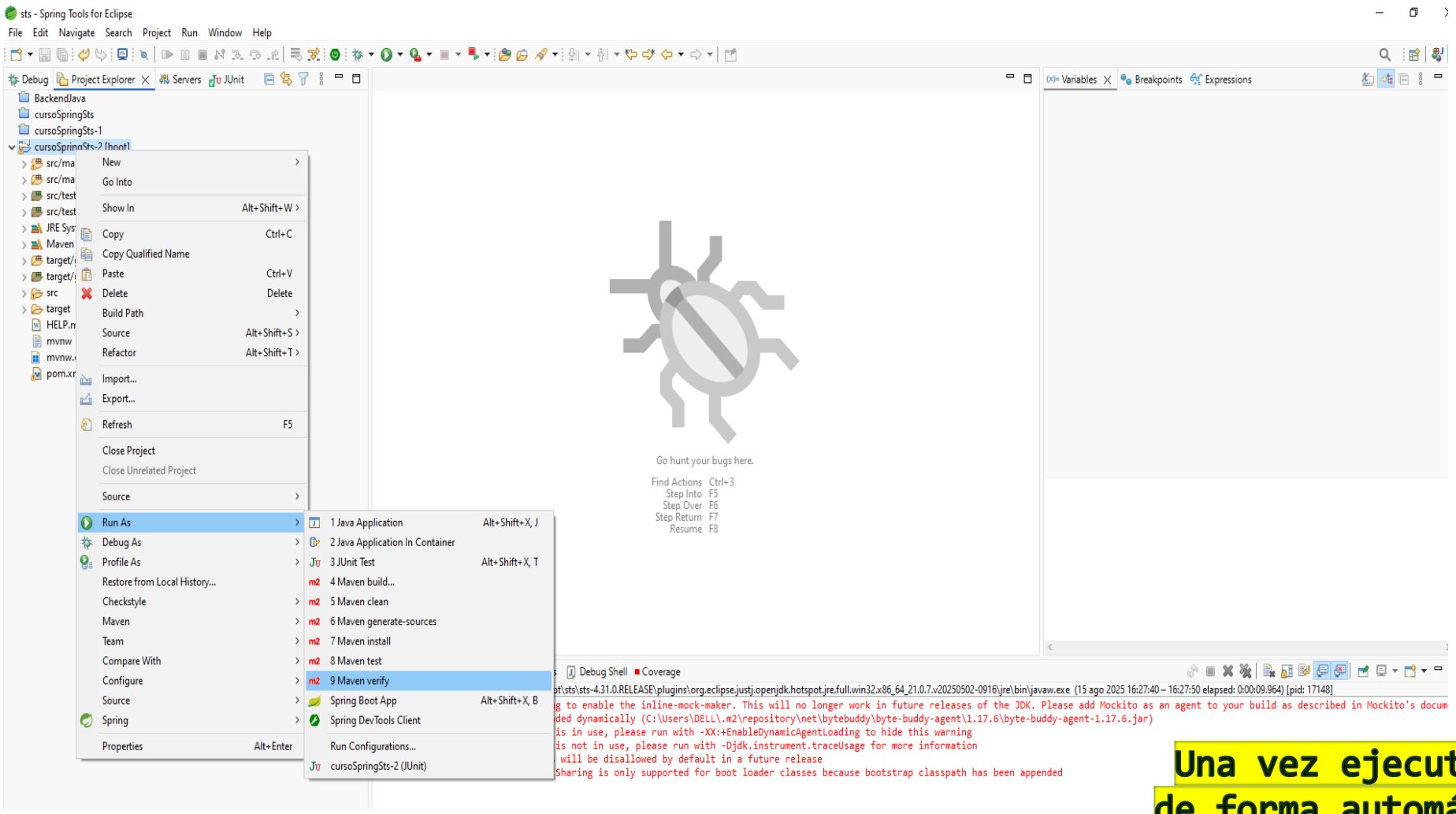
Configuración de Jacoco Para reporte de cobertura de código probado

Para la configuración de dependencias de Jacoco se validará que se tenga registrado El plugin en el POM .

```
<!-- JaCoCo para cobertura -->
<plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.12</version>
        <executions>
                <execution>
                <id>prepare-agent</id>
                <goals>
                <goal>prepare-agent</goal>
                </goals>
                </execution>
                <execution>
                <id>report</id>
                <phase>verify</phase>
                <goals>
                <goal>report</goal>
                </goals>
                </execution>
        </executions>
</plugin>
```

Con esta configuración estamos listos para realizar las pruebas de cobertura.

Crear nuestro reporte de cobertura de pruebas realizadas

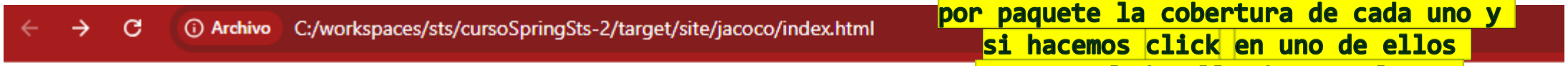


Una vez ejecutado, se genera de forma automática el reporte de cobertura de código, que a continuación veremos.

Crear nuestro reporte de cobertura de pruebas realizadas en esta ruta /cursoSpringSts-2/target/site/jacoco

Nombre	Fecha de modificación	Tipo	Tamaño
com.curso.spring	15/08/2025 04:51 p. m.	Carpeta de archivos	
com.curso.spring.controller	14/08/2025 12:00 p. m.	Carpeta de archivos	
com.curso.spring.exceptions	14/08/2025 12:00 p. m.	Carpeta de archivos	
com.curso.spring.services	14/08/2025 12:00 p. m.	Carpeta de archivos	
com.curso.spring.vo	14/08/2025 12:00 p. m.	Carpeta de archivos	
jacoco-resources	14/08/2025 12:00 p. m.	Carpeta de archivos	
index	15/08/2025 04:52 p. m.	Chrome HTML Do...	6 KB
jacoco	15/08/2025 04:52 p. m.	Archivo de valores...	1 KB
jacoco	15/08/2025 04:52 p. m.	Archivo XML	17 KB
jacoco-sessions	15/08/2025 04:52 p. m.	Chrome HTML Do...	845 KB

Abriendo el archivo Index.html, veremos por paquete la cobertura de cada uno y si hacemos click en uno de ellos veremos el detalle de sus clases.



cursoSpringSts-2

cursoSpringSts-2

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.curso.spring.services		3%		0%	9	10	21	22	6	7	0	1
com.curso.spring.controller		89%		n/a	0	6	8	81	0	6	0	1
com.curso.spring.vo		50%		n/a	1	2	1	3	1	2	0	1
com.curso.spring		37%		n/a	1	2	2	3	1	2	0	1
com.curso.spring.exceptions		100%		n/a	0	2	0	4	0	2	0	2
Total	136 of 462	70%	6 of 6	0%	11	22	32	113	8	19	0	6

Validar deuda técnica

- Ingresaremos la configuración en nuestro ya conocido archivo POM.xml

```
<!-- Sonar -->
<plugin>
        <groupId>org.sonarsource.scanner.maven</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
        <version>3.9.1.2184</version>
</plugin>
```

- Crearemos el archivo sonar-project.properties en al ruta \src\main\resources
- Añadimos la siguiente configuración

```
sonar.projectKey=cursoSpring-2
sonar.sources=src
sonar.java.binaries=target/classes
sonar.login=sqp_02ef4e693ccc4decd41c2d74f87203179f63c97c
```

- Utilizando la siguiente línea de comando para ejecutar la verificación de código, en la ruta raíz del proyecto, donde se haya el POM.

```
mvn clean verify sonar:sonar -Dsonar.projectKey=cursoSpring-2 -Dsonar.projectName='cursoSpring-2' -Dsonar.host.url=http://localhost:9000 -Dsonar.token=sqp_02ef4e693ccc4decd41c2d74f87203179f63c97c
```

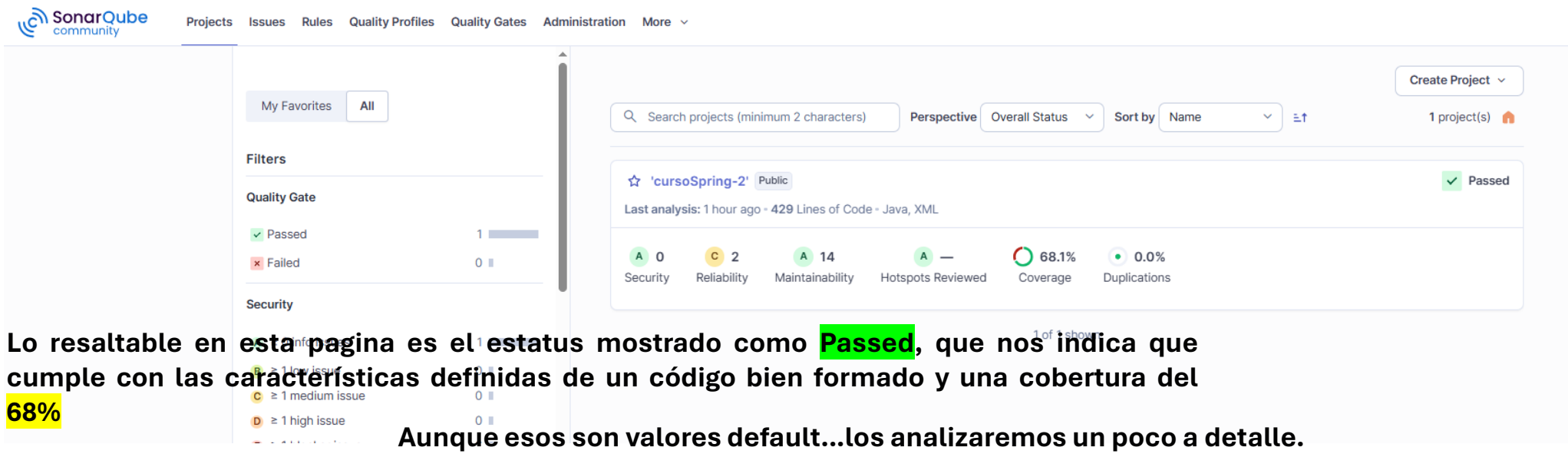
(Sustituir por token generado en servidor sonar, ver el video de instalación de sonarQube)

Ejecución validación sonar

Una vez terminada la ejecución del sonar.

```
[INFO] Available processors: 4
[INFO] Using 4 threads for analysis.
[INFO] Start fetching files for the text and secrets analysis
[INFO] Using JGit to retrieve untracked files
[WARNING] Retrieving only language associated files, make sure to run the analysis inside a git repository to make use of inclusions specified via "sonar.text.inclusions"
[INFO] Starting the text and secrets analysis
[INFO] 12 source files to be analyzed for the text and secrets analysis
[INFO] 12/12 source files have been analyzed for the text and secrets analysis
[INFO] Sensor TextAndSecretsSensor [text] (done) | time=1525ms
[INFO] ------------- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=3ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=79ms
[INFO] ------------- Gather SCA dependencies on project
[INFO] Dependency analysis skipped
[INFO] SCM Publisher No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
[INFO] CPD Executor 6 files had no CPD blocks
[INFO] CPD Executor Calculating CPD for 3 files
[INFO] CPD Executor CPD calculation finished (done) | time=43ms
[INFO] Analysis report generated in 223ms, dir size=303.3 kB
[INFO] Analysis report compressed in 268ms, zip size=56.6 kB
[INFO] Analysis report uploaded in 89ms
[INFO] ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=cursoSpring-2
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=e54b5a7b-58ed-4dea-b762-5e1ae2e6e302
[INFO] Analysis total time: 13.607 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:05 min
[INFO] Finished at: 2025-08-16T15:13:18-06:00
[INFO] ------------------------------------------------------------------------

\workspaces\sts\cursoSpringSts-2>
```

Nos mostrara la siguiente información en nuestro servidor sonarqube.



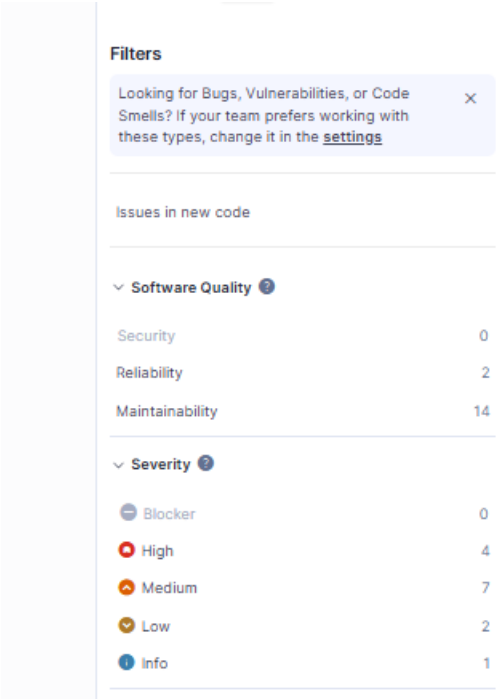
Lo resaltable en esta pagina es el estatus mostrado como Passed, que nos indica que cumple con las características definidas de un código bien formado y una cobertura del 68%

Aunque esos son valores default…los analizaremos un poco a detalle.

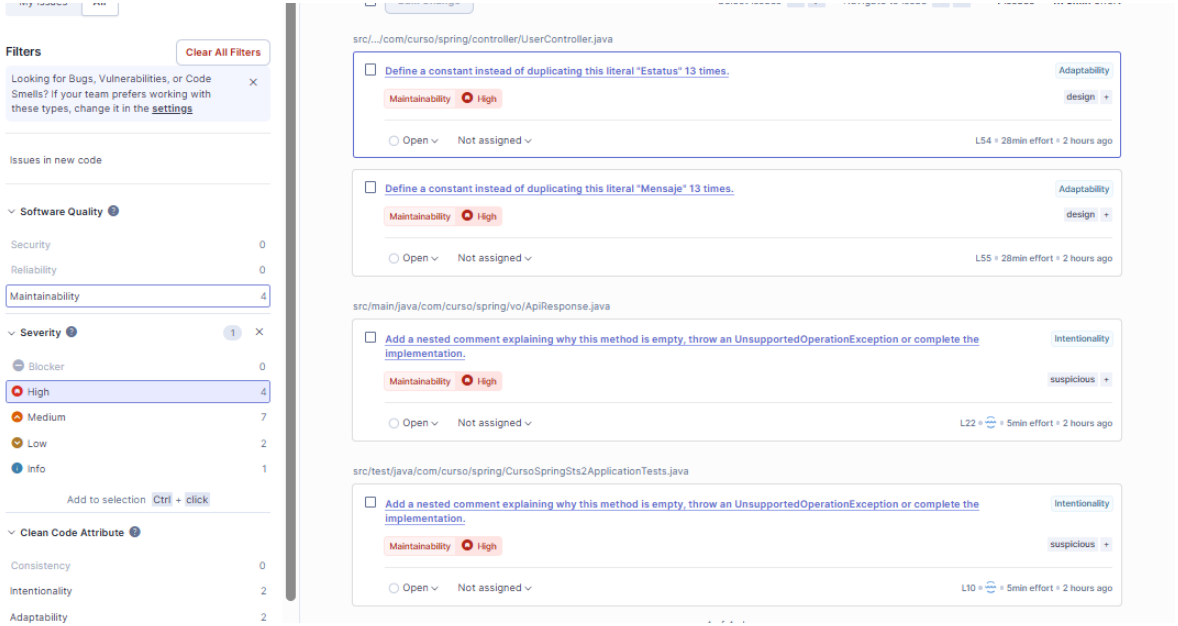
Ejecución validación sonar

Revisando a fondo los datos mostrados vemos lo siguiente

Revisando el menú de Issues



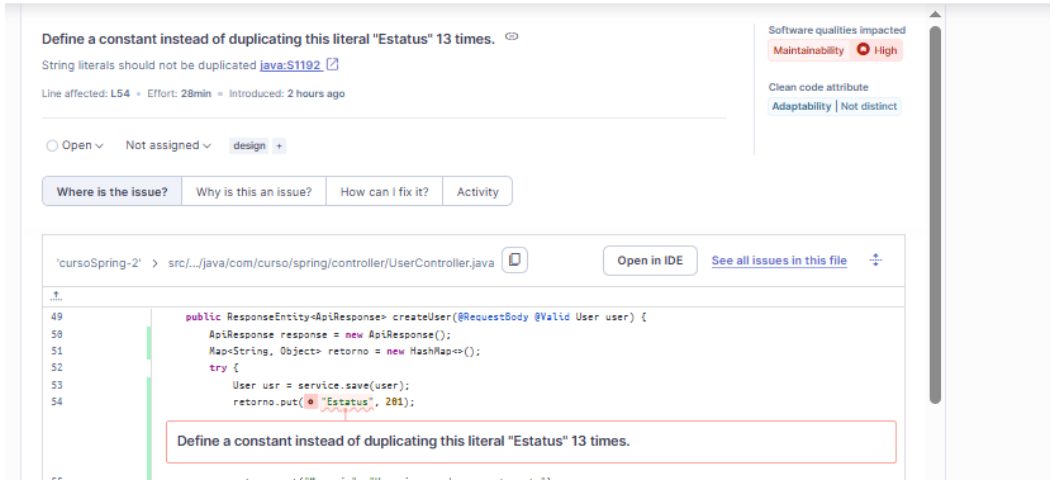
Haciendo click en la etiqueta de la severidad del Issue, nos mostrara que archivo y líneas contienen esos iusses, ejemplo haciendo click en severidad High



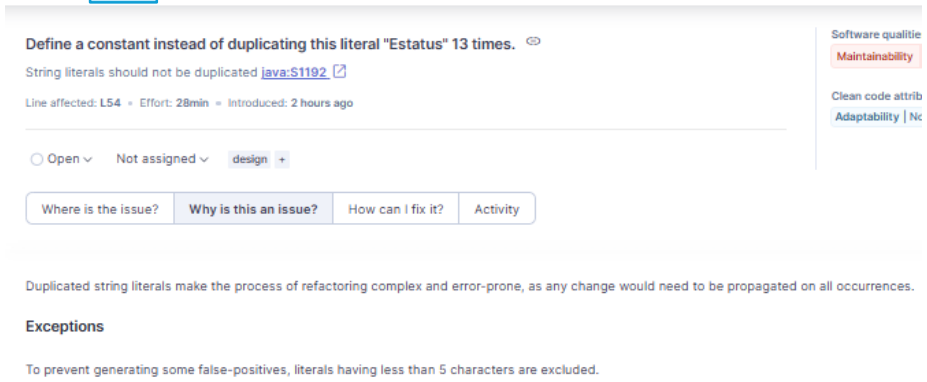
Ejecución validación sonar

Seleccionamos el primer Issue, 1 nos indica línea, así como el 2 porqué del issue y 3 como corregirlo.

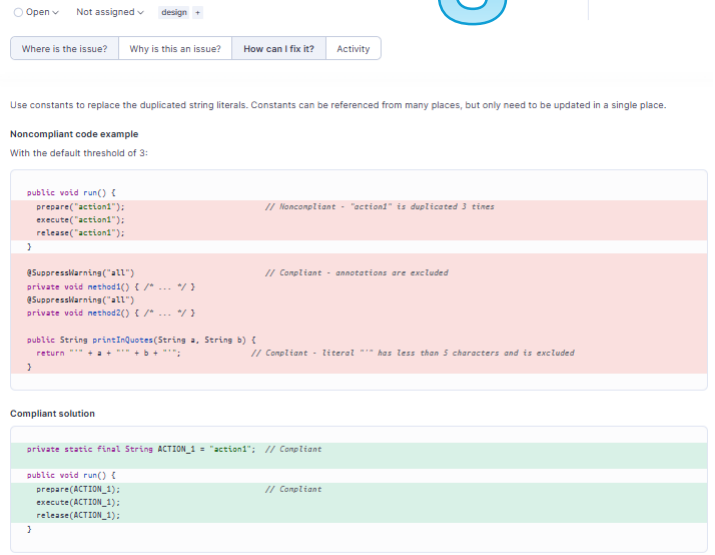
1



2



3



Ejecución validación sonar

Con cada uno de los Issues se debe corregir para que nuestro código quede 100% limpio y profesional.

Cuando se haya realizado todos estos cambios, se debe actualizar el proyecto y volver a ejecutar las pruebas unitarias, reportye de cobertura, reporte de sonar para garantizar que todo nuestro código realizado esta cumpliendo con nuestras mejores practicas de desarrollo.

En nuestro siguiente Curso, veremos como realizar un pipeline automatizado para que todo este proceso de modificación de código, pruebas unitarias, coberturas y sonar, sea realizado de forma automática, permitiendo dar a nuestro equipo de trabajo un ambiente de desarrollo estable y dinámico para integra cambios a nuestro desarrollo y generar despliegues cuando se cumplan las reglas y mejores prácticas del desarrollo.

No olvides dejar tu like y regístrate para que te avisen cuando se integren nuevos videos.

Hasta la próxima y a continuar practicando.....