



Pruebas unitarias con JUnit y Mockito

Indice

- ¿Qué son las pruebas unitarias?
- Principio FIRST
- La AAA
- Proceso de pruebas unitarias
- Casos de pruebas
- Beneficios de las pruebas unitarias
- Junit
- Mockito
- Pasos a seguir
- Creando prueba unitaria a Service

¿Qué son las pruebas unitarias?

- Las pruebas unitarias o *unit testing* son una forma de comprobar que un fragmento de código funciona correctamente.
- Consisten en aislar una parte de código y comprobar que funciona a la perfección.
- Se basan en pequeños *test* que validan el comportamiento de un objeto y la lógica.
- Suelen realizarse durante la fase de desarrollo de aplicaciones de software y normalmente las llevan a cabo los desarrolladores.
- Con las pruebas unitarias se detectan errores que no se podrían detectar hasta fases más avanzadas.

Principio FIRST

- Fast (Rápida): las pruebas se deben ejecutar rápidamente.
- Isolated (Aisladas): las pruebas deben ser independientes unas de otras y deben poder ejecutarse en cualquier orden sin que eso afecte el resultado.
- Repeatable (Repetible): las pruebas se deben de poder repetir en cualquier ambiente obteniendo el mismo resultado en cada uno.
- Self-validating (Autoevaluación): las pruebas deben tener una validación que de forma sencilla indiquen si la prueba resulto valida o fallida.
- Timely (En tiempo): Las pruebas son creadas al mismo tiempo que el software que se esta creando.

La AAA



- Para llevar a cabo buenas pruebas unitarias, deben estar estructuradas siguiendo las tres A's del unit testing. Se trata de un concepto que describe un proceso de tres pasos:
- Arrange (organizar): en esta parte se deben establecer las condiciones iniciales para realizarla y el resultado esperado.
- Act (ejecutar): es la parte de ejecución del fragmento de código de la prueba.
- Assert (validación): se realiza la comprobación para verificar que el resultado obtenido coincide con el esperado.

Proceso de pruebas unitarias

- El proceso de pruebas unitarias puede realizarse de manera manual, aunque lo más común es automatizar el procedimiento con herramientas.
- Ejemplos de herramientas que ayudan con la elaboración de las pruebas:
- xUnit: herramienta de pruebas unitarias para el framework .NET
- JUnit: es un conjunto de librerías para realizar pruebas unitarias de aplicaciones Java.
- NUnit: herramienta para el soporte para una amplia gama de plataformas .NET
- PHPUnit: entorno de pruebas unitarias en el lenguaje de programación PHP.

Casos de pruebas

- Al ser el programador
 - quien escribe el código
 - quien escribe las pruebas unitarias
- debe ser precavido
 - al confiar en que la unidad de código funciona correctamente
 - escriba pruebas unitarias menos exigentes que no contemple casos que pueden fallar.
- Los casos de pruebas que se debe de considerar son:
- El happy path.
- Caminos alternos.
- Excepciones.



Beneficios de las pruebas unitarias

- Las pruebas unitarias tienen muchas ventajas:
- Permite ahorrar tiempo en las pruebas de regresión.
- Menos errores que se escapan a la naturaleza de la lógica.
- Tener más recursos valiosos:
 - La creatividad y la innovación.
 - Mejores soluciones.
 - Ser más productivo.
- Documentación actualizada.
- Reducir la carga de tareas al personal de soporte.

JUnit

- JUnit es un framework de código abierto que nos permite escribir y ejecutar pruebas unitarias en Java. JUnit permite ejecutar pruebas automatizadas repetibles.
- JUnit ha evolucionado con el tiempo y el cambio principal a tener en cuenta es la introducción de anotaciones que se introdujeron en la versión JUnit 4.
- JUnit permite ejecutar todas las pruebas automáticamente o una selección específica y luego informar el resultado.
- Por lo general viene como una extensión o plugin del IDE y se puede utilizar en línea de comandos.

Mockito

- Mockito es un framework de código abierto que nos permite la creación de objetos simulados con el propósito de realizar pruebas unitarias en Java.
- Mockito permite crear test doubles o fakes fácilmente.
 - Test doubles: los test doubles son un término genérico que hace referencia a cualquier caso en el que se reemplaza un objeto de producción con otro con el único objeto de probar el código.
- Mockito le proporciona a una clase que se probará las instancias de sus clases colaboradoras. Permitiendo personalizar fácilmente el comportamiento de las clases colaboradoras a las necesidades de la prueba.

Pasos a seguir

- Configurar el POM para utilizar Junit y Mockito

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-test</artifactId>  
  <scope>test</scope>  
</dependency>
```

Configurar POM para usar Jacoco

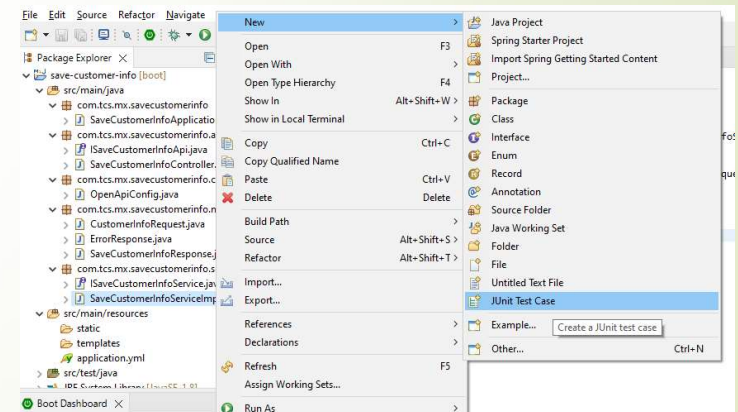
- Configurar el POM para utilizar Jacoco y ver la cobertura de las pruebas unitarias.

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>${jacoco.version}</version>
  <configuration>
    <skip>${maven.test.skip}</skip>
    <destFile>${basedir}/target/coverage-reports/jacoco-unit.exec</destFile>
    <dataFile>${basedir}/target/coverage-reports/jacoco-unit.exec</dataFile>
    <output>file</output>
    <append>true</append>
    <excludes>
      <exclude>**/config/*Config.java</exclude>
      <exclude>**/models/dto/*</exclude>
      <exclude>**/*Test.*</exclude>
    </excludes>
  </configuration>
  <executions>
    <execution>
      <id>jacoco-initialize</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
      <phase>test-compile</phase>
    </execution>
    <execution>
      <id>jacoco-site</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Creando prueba unitaria a Service

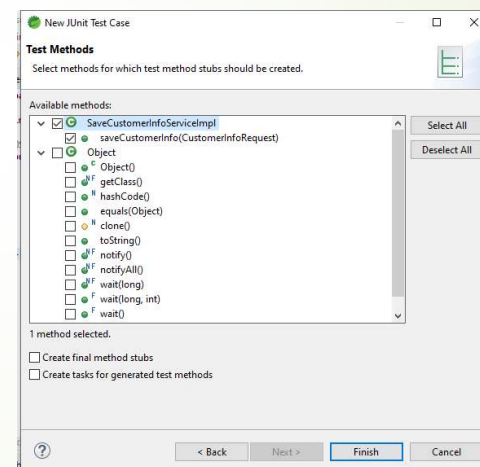
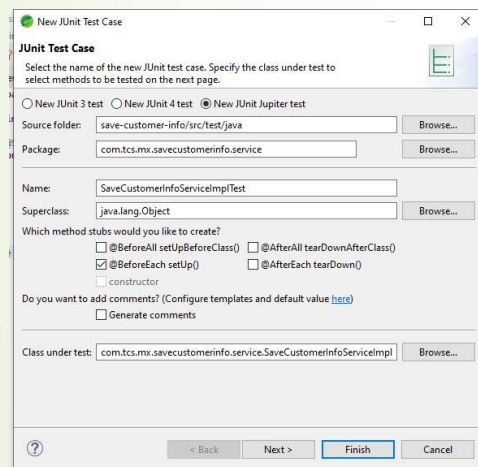
- Lo primero que realizaremos es la prueba a la clase `SaveCustomerInfoServiceImpl`.

```
1 package com.tcs.mx.savecustomerinfo.service;  
2  
3 import org.springframework.stereotype.Service;  
4  
5 @Service  
6 public class SaveCustomerInfoServiceImpl implements ISaveCustomerInfoService {  
7  
8     @Override  
9     public SaveCustomerInfoResponse saveCustomerInfo(CustomerInfoRequest customerInfo) {  
10         // TODO Auto-generated method stub  
11         return null;  
12     }  
13  
14 }  
15  
16  
17  
18
```



Utilizando el wizard para creación

- Utilizar el paso a paso para crear una prueba unitaria.



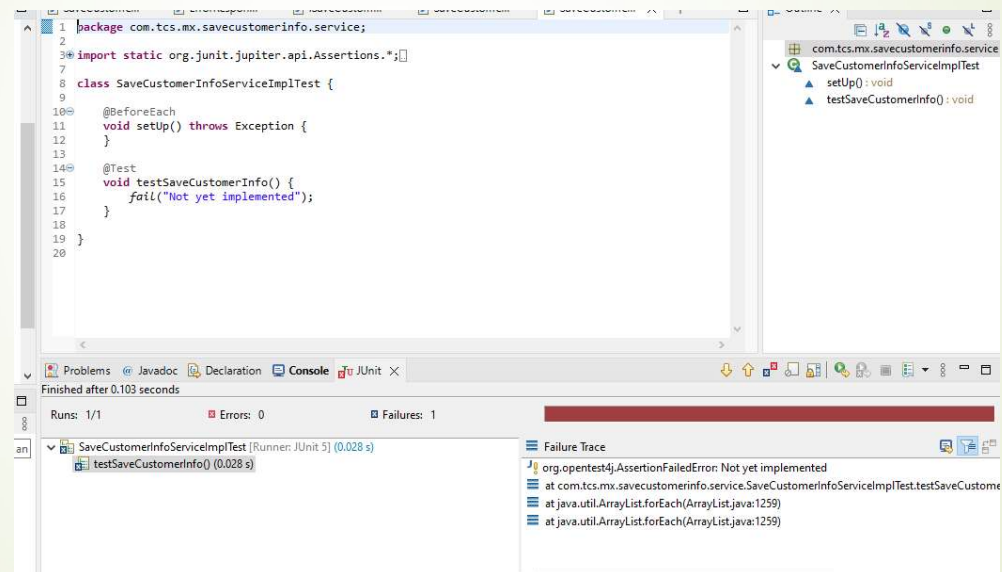
Revisando los elementos de la prueba unitaria

- Revisando los elementos de la prueba unitaria.

```
1 package com.tcs.mx.savecustomerinfo.service;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class SaveCustomerInfoServiceImplTest {
6
7     @BeforeEach
8     void setUp() throws Exception {
9     }
10
11     @Test
12     void testSaveCustomerInfo() {
13         fail("Not yet implemented");
14     }
15 }
16
17
18
19
20
```

Primera ejecución de la prueba unitaria

- ▶ Ejecutar la prueba que debe fallar.



The screenshot shows an IDE with a Java unit test file named `SaveCustomerInfoServiceImplTest`. The code defines a test class with a `setUp()` method and a `testSaveCustomerInfo()` method that calls `fail("Not yet implemented")`. The test is executed, and the results are shown in the bottom panels. The `JUnit` panel indicates 1 failure. The `Failure Trace` panel shows the stack trace for the failure.

```
1 package com.tcs.mx.savecustomerinfo.service;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class SaveCustomerInfoServiceImplTest {
6
7     @BeforeEach
8     void setUp() throws Exception {
9     }
10
11     @Test
12     void testSaveCustomerInfo() {
13         fail("Not yet implemented");
14     }
15 }
```

JUnit Results: Finished after 0.103 seconds. Runs: 1/1. Errors: 0. Failures: 1.

Failure Trace:

- org.opentest4j.AssertionFailedError: Not yet implemented
- at com.tcs.mx.savecustomerinfo.service.SaveCustomerInfoServiceImplTest.testSaveCustomerInfo(SaveCustomerInfoServiceImplTest.java:1259)
- at java.util.ArrayList.forEach(ArrayList.java:1259)

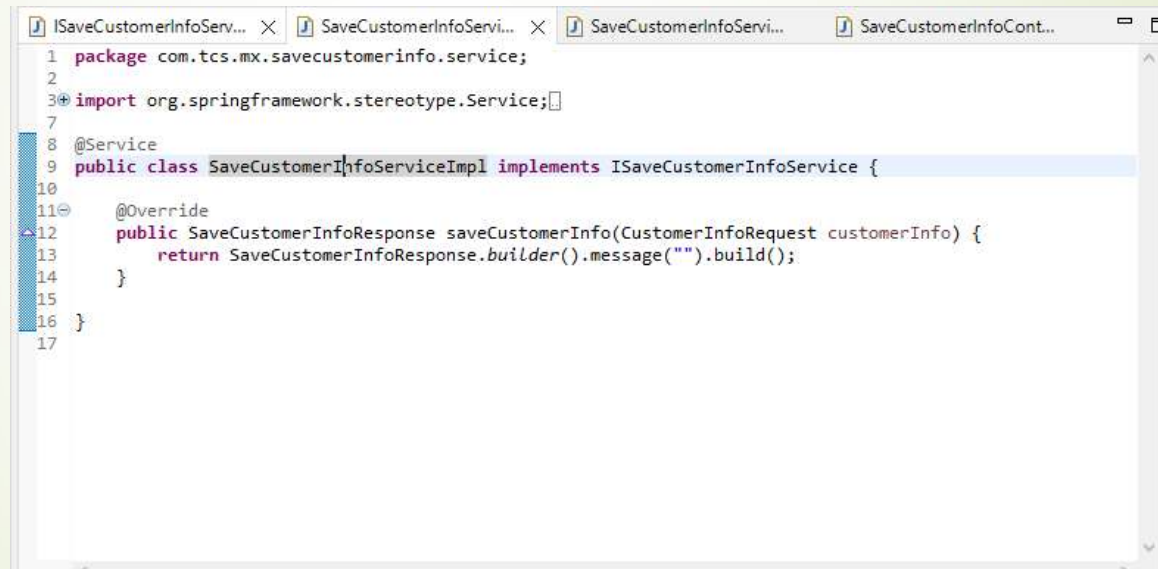
Escribir la prueba unitaria para el happy path

- Revisando las AAA's para la prueba unitaria:

```
4 @SpringBootTest
5 class SaveCustomerInfoServiceImplTest {
6
7     private static final String SUCCESSFUL_CLIENT_REGISTRATION_MSG = "Successful client registration";
8
9     @InjectMocks
10    private SaveCustomerInfoServiceImpl service;
11
12    private CustomerInfoRequest request;
13    private SaveCustomerInfoResponse mockResponse;
14
15    @BeforeEach
16    void setUp() throws Exception {
17        request = CustomerInfoRequest.builder()
18            .customerId("09507812-2")
19            .customerName("Luis Sanchez").build();
20
21        mockResponse = SaveCustomerInfoResponse.builder()
22            .message(SUCCESSFUL_CLIENT_REGISTRATION_MSG).build();
23    }
24
25    @DisplayName("Prueba metodo SaveCustomerInfo cuando resultado es OK")
26    @Test
27    void testSaveCustomerInfo() {
28
29        SaveCustomerInfoResponse saveCustomerInfoResponse = service.saveCustomerInfo(request);
30
31        assertEquals(mockResponse.getMessage(), saveCustomerInfoResponse.getMessage(),
32            "Error al validar respuesta message");
33    }
34 }
35 }
```

Escribir el código del componente

- Escribir el código del método al que se le realizará la prueba unitaria.



```
1 package com.tcs.mx.savecustomerinfo.service;
2
3 import org.springframework.stereotype.Service;
4
5
6
7
8 @Service
9 public class SaveCustomerInfoServiceImpl implements ISaveCustomerInfoService {
10
11     @Override
12     public SaveCustomerInfoResponse saveCustomerInfo(CustomerInfoRequest customerInfo) {
13         return SaveCustomerInfoResponse.builder().message("").build();
14     }
15
16 }
17
```

Ejecutando prueba y validación

- Se revisa el resultado de la ejecución de la prueba:

The screenshot shows an IDE with a Java test class and its execution results. The test class is `SaveCustomerInfoServiceImplTest` and the test method is `testSaveCustomerInfo()`. The test method is annotated with `@Test` and `@DisplayName("Prueba metodo SaveCustomerInfo cuando resultado es OK")`. The test method calls `service.saveCustomerInfo(request)` and asserts that the response message is `SUCCESSFUL_CLIENT_REGISTRATION_MSG`. The execution results show that the test failed. The failure trace indicates that the assertion failed because the response message was not as expected.

```
22 private CustomerInfoRequest request;
23 private SaveCustomerInfoResponse mockResponse;
24
25 @BeforeEach
26 void setUp() throws Exception {
27     request = CustomerInfoRequest.builder()
28         .customerId("09507812-2")
29         .customerName("Luis Sanchez").build();
30
31     mockResponse = SaveCustomerInfoResponse.builder()
32         .message(SUCCESSFUL_CLIENT_REGISTRATION_MSG).build();
33 }
34
35 @DisplayName("Prueba metodo SaveCustomerInfo cuando resultado es OK")
36 @Test
37 void testSaveCustomerInfo() {
38
39     SaveCustomerInfoResponse saveCustomerInfoResponse = service.saveCustomerInfo(request);
40
41     assertEquals(mockResponse.getMessage(), saveCustomerInfoResponse.getMessage(),
42         "Error al validar respuesta message");
43 }
44
```

Problems @ Javadoc Declaration Console JUnit

Finished after 3.753 seconds

Runs: 1/1 Errors: 0 Failures: 1

SaveCustomerInfoServiceImplTest [Runner: JUnit 5] (0.076 s)

Prueba metodo SaveCustomerInfo cuando resultado es OK (0.076 s)

Failure Trace

```
org.opentest4j.AssertionFailedError: Error al validar respuesta message ==> expected: <Successful Client Registration> but was: <Error al validar respuesta message>
at com.tcs.mx.savecustomerinfo.service.SaveCustomerInfoServiceImplTest.testSaveCustomerInfo(SaveCustomerInfoServiceImplTest.java:42)
at java.util.ArrayList.forEach(ArrayList.java:1259)
at java.util.ArrayList.forEach(ArrayList.java:1259)
```

Modificando código de service y ejecutando prueba de nuevo

- Se modifica el código del componente en el service y se ejecuta la prueba de nuevo.

```
1 package com.tcs.mx.savecustomerinfo.service;
2
3 import java.time.LocalDateTime;
4
5
6
7
8
9
10 @Service
11 public class SaveCustomerInfoServiceImpl implements ISaveCustomerInfoService {
12
13     → private static final String SUCCESSFUL_CLIENT_REGISTRATION_MSG = "Successful client registration";
14
15
16     @Override
17     public SaveCustomerInfoResponse saveCustomerInfo(CustomerInfoRequest customerInfo) {
18         return SaveCustomerInfoResponse.builder()
19             .message(SUCCESSFUL_CLIENT_REGISTRATION_MSG)
20             .timestamp(LocalDateTime.now())
21             .build();
22     }
23 }
24
```

```
1 package com.tcs.mx.savecustomerinfo.service;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6
7
8
9
10
11
12
13
14 @SpringBootTest
15 class SaveCustomerInfoServiceImplTest {
16
17     private static final String SUCCESSFUL_CLIENT_REGISTRATION_MSG = "Successful client registration";
18
19     @InjectMocks
20     private SaveCustomerInfoServiceImpl service;
21
22     private CustomerInfoRequest request;
23     private SaveCustomerInfoResponse mockResponse;
24
25     @BeforeEach
26     void setUp() throws Exception {
27         request = CustomerInfoRequest.builder()
28             .customerId("09507812-2")
29             .customerName("Luis Sanchez").build();
30
31         mockResponse = SaveCustomerInfoResponse.builder()
32             .message(SUCCESSFUL_CLIENT_REGISTRATION_MSG).build();
33     }
34
35     @Test
36     void testSaveCustomerInfo() {
37         SaveCustomerInfoResponse response = service.saveCustomerInfo(request);
38         assertEquals(SUCCESSFUL_CLIENT_REGISTRATION_MSG, response.getMessage());
39         assertNotNull(response.getTimestamp());
40     }
41 }
42
```

Problems | Javadoc | Declaration | Console | JUnit

Finished after 3.75 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

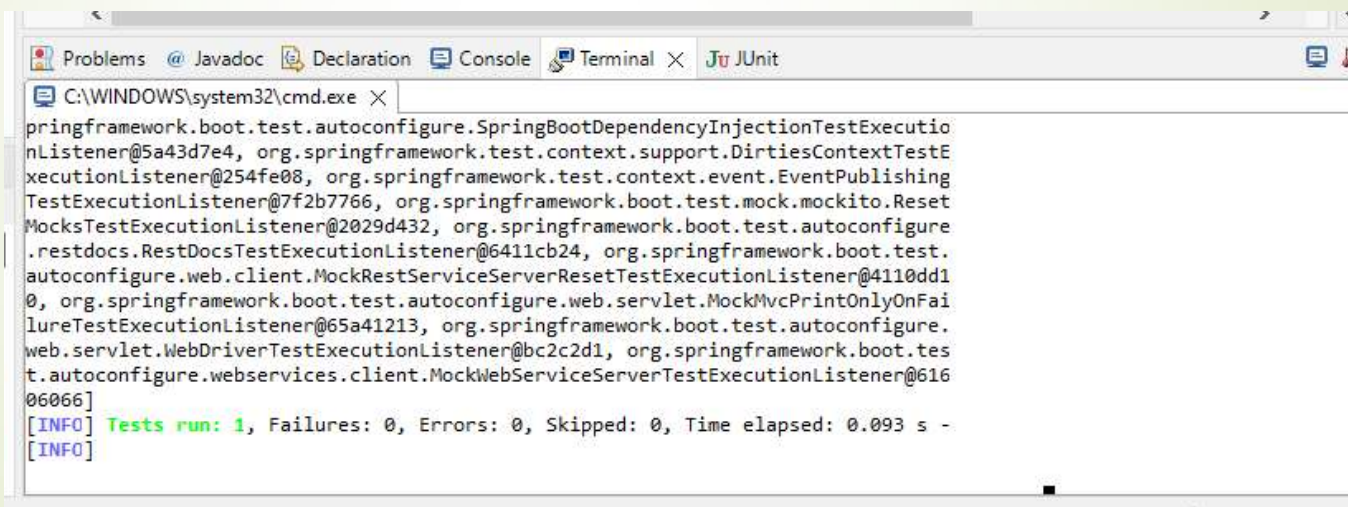
SaveCustomerInfoServiceImplTest [Runner: JUnit 5] (0.046 s)

→ Prueba metodo SaveCustomerInfo cuando resultado es OK (0.046 s)

Failure Trace

Ejecutar comando para generar reporte en sonarqube

- Cobertura: Indicador que nos dirá cuantas líneas de nuestro código fueron ejecutadas durante la realización de las pruebas unitarias.
- Ejecutar comando mvn sonar:sonar para generar reportes de cobertura.



The screenshot shows an IDE window with a terminal tab. The terminal output displays a list of test execution listeners and a summary of the test results. The summary indicates that 1 test was run successfully with no failures, errors, or skips, and the execution time was 0.093 seconds.

```
C:\WINDOWS\system32\cmd.exe X
pringframework.boot.test.autoconfigure.SpringBootDependencyInjectionTestExecutio
nListener@5a43d7e4, org.springframework.test.context.support.DirtiesContextTestE
xecutionListener@254fe08, org.springframework.test.context.event.EventPublishing
TestExecutionListener@7f2b7766, org.springframework.boot.test.mock.mockito.Reset
MocksTestExecutionListener@2029d432, org.springframework.boot.test.autoconfigure
.restdocs.RestDocsTestExecutionListener@6411cb24, org.springframework.boot.test.
autoconfigure.web.client.MockRestServiceServerResetTestExecutionListener@4110dd1
0, org.springframework.boot.test.autoconfigure.web.servlet.MockMvcPrintOnlyOnFai
lureTestExecutionListener@65a41213, org.springframework.boot.test.autoconfigure.
web.servlet.WebDriverTestExecutionListener@bc2c2d1, org.springframework.boot.tes
t.autoconfigure.webservices.client.MockWebServiceServerTestExecutionListener@616
06066
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.093 s -
[INFO]
```

Revisar estadísticas en sonarqube

- Se revisa la pagina principal del proyecto

The screenshot displays the SonarQube web interface in a browser. The address bar shows 'localhost:9000/projects'. The navigation bar includes links for 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar is present on the right. The main content area shows the 'Overall Status' perspective for a project named 'save-customer-info', which is marked as 'Passed'. The statistics are as follows:

Metric	Value	Grade
Bugs	1	C
Vulnerabilities	0	A
Code Smells	1	A
Coverage	51.3%	
Duplications	0.0%	

The last analysis was performed on March 22, 2022, at 2:33 AM. The project has 256 files (Java, XML). A warning message at the bottom states: 'Embedded database should be used for evaluation purposes only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.'

Revisar cobertura de clase Service

- Se revisa la cobertura de la clase Service

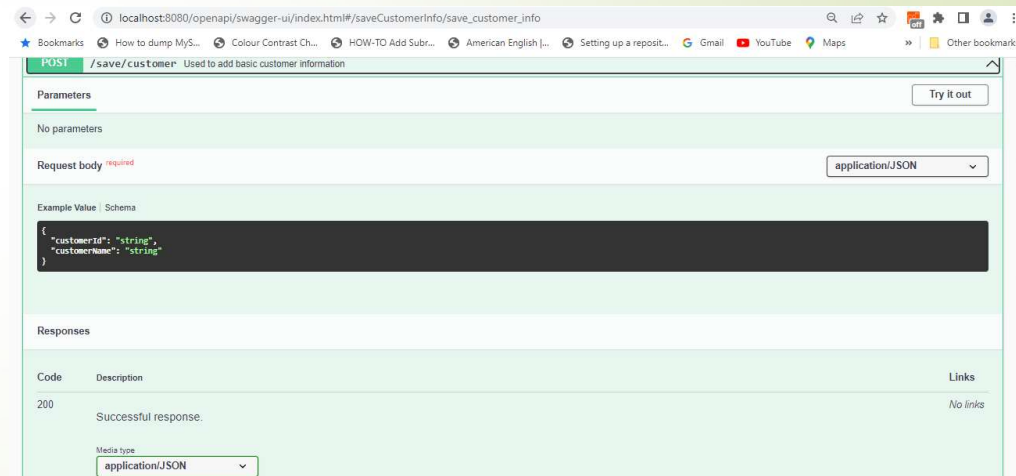
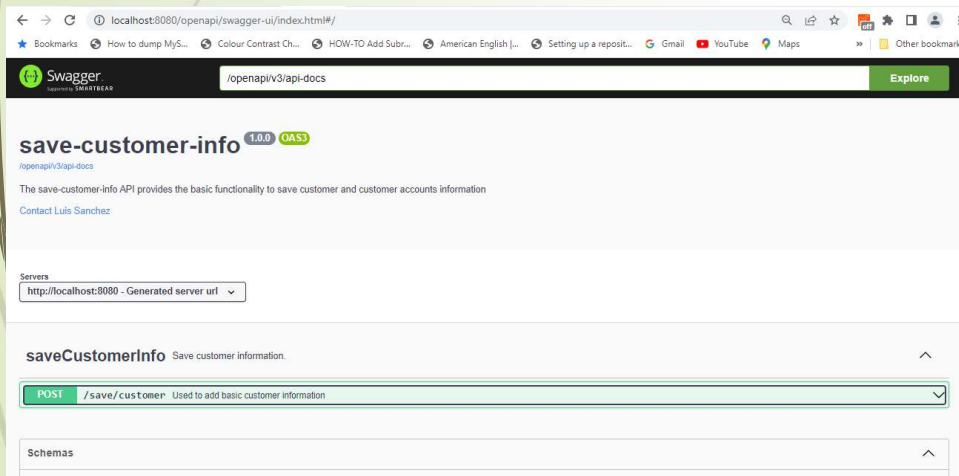
The screenshot displays the SonarQube web interface. The browser address bar shows a URL for component measures. The SonarQube header includes navigation links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. The main content area is for the project 'save-customer-info' on the 'master' branch. A sidebar on the left shows the 'Coverage' section with a table of metrics:

Metric	Value
Coverage	51.3%
Lines to Cover	35
Uncovered Lines	15
Line Coverage	57.1%
Conditions to Cover	4
Uncovered Conditions	4
Condition Coverage	0.0%

The main panel shows the source code for the file 'src/main/java/com/tcs/mx/savecustomerinfo/service/SaveCustomerInfoServiceImpl.java'. The code is a Java class implementing the 'ISaveCustomerInfoService' interface. The coverage status is shown as 24 Lines, 0 Issues, and 100% Coverage. The code includes package declarations, imports for 'java.time.LocalDateTime', 'org.springframework.stereotype.Service', and two DTOs from 'com.tcs.mx.savecustomerinfo.model.dto'. The class contains a service method 'saveCustomerInfo' that returns a 'SaveCustomerInfoResponse' object.

Validando proyecto con API

- ▶ Ejecutando endpoint desde API de proyecto generado.



Ejecutando endpoint desde API

➤ Se ejecuta el endpoint:

The image shows the Swagger UI for the endpoint `POST /save/customer`. The description is "Used to add basic customer information". There are no parameters. The request body is required and set to `application/json`. The body contains the following JSON:

```
{  "customerId": "09507812-3",  "customerName": "Luis Sanchez M"}
```

At the bottom, there is a blue "Execute" button.

The image shows the Swagger UI for the endpoint `POST /save/customer` after execution. It displays the "Responses" section with a 200 status code. The response body is:

```
{  "message": "Successful client registration",  "timestamp": "2022-03-22T02:18:48.419"}
```

The response headers are:

```
connection: keep-alive
content-type: application/json
date: Tue, 22 Mar 2022 02:18:48 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```



Gracias

Luis Sánchez Martínez