MEMORIA PROYECTO BIBLIOTECA

ISO 2

ENLACES:

Github:
https://github.com/mbk5574/Grupo1_ISOII

SonarCloud:
https://sonarcloud.io/project/overview?id=mbk5574_Grupo1_ISOII

BASE DE DATOS

Usuario: derbyuser

Contraseña: password

AUTORES:

Mohammed Bannay Khyy

Mohammed.bannay@alu.uclm.es

Álvaro Parra Rufo

Alvaro.parra2@alu.uclm.es

Andrea Vicario Dorado

andrea.vicario@alu.uclm.es

Badr Eddine Allaoui Zoggagh

Badreddine.allaoui@alu.uclm.es

Contenido

1. Introducción 2
2. Requisitos funcionales y no funcionales 2
   2.1 Requisitos funcionales 2
   2.2 Requisitos no funcionales 3
3. Plan de gestión de configuración 3
   3.1 Objetivos: 3
   3.2 Identificación de la configuración: 3
   3.3 Control de versiones 4
   3.4 Gestión de cambios 4
   3.5 Construcción y despliegue 4
   3.6 Auditoría y trazabilidad 4
   3.7 Respaldo y restauración 5
   3.8 Documentación 5
   4. Control de versiones 5
   4.1 GitHub 5
      4.1.1 main 5
      4.1.2 Dev 5
      4.1.3 funcionalidad_RegistrarUsuario 5
      4.1.4 funcionalidad_RealizarReserva 5
      4.1.5 funcionalidad_aplicarPenalización 5
      4.1.6 funcionalidad_comprobarPenalizacion 5
      4.1.7 funcionalidad_DevoluciónEjemplar: 6
      4.1.8 funcionalidad_PréstamoEjemplar 6
5. Organización del proyecto 6
6. Sprints 8
7. Calidad 11
   7.1 Calidad de producto 11
   7.2 Configuración de SonarCloud 11
   7.3 Análisis continuo de Sonar 12
   7.4 Despliegue continuo 13
8. Testing 14
   8.1 Ejemplos de testing 15
9. Mantenimiento 19

1. Introducción

En este trabajo se presenta tanto el diseño como la implementación de un sistema web para la gestión de una biblioteca. El objetivo de este trabajo es facilitar la gestión de los ejemplares, los usuarios, y los préstamos de una biblioteca, así como ofrecer una interfaz sencilla y funcional a los futuros usuarios que harán uso de la página web. Se mantiene un diseño claro y conciso para que el administrador de la biblioteca pueda hacer uso de todas las funcionalidades que le proporciona el sistema para mantener un seguimiento de todos los recursos de la biblioteca y evitar que puedan ser objeto de un mal uso gracias al seguimiento (ejemplo: prestamos que no se devuelven, usuario con 30 ejemplares prestados, etc).

2. Requisitos funcionales y no funcionales

2.1 Requisitos funcionales

Un requisito funcional es una declaración de lo que debe hacer un sistema, esto es, son los servicios que ofrece.

Algunos requisitos funcionales son:

Gestión de usuarios
Registrar usuario

Gestión de Ejemplares
Añadir un nuevo ejemplar
Borrar un ejemplar

Gestión de Títulos
Dar de alta un nuevo título
Actualizar un título
Borrar un título

Gestión de Reservas
Realizar reserva

Gestión de Préstamos
Dar de alta un préstamo

Multas y penalizaciones

2.2 Requisitos no funcionales

Un requisito no funcional es una característica o restricción del sistema.
Algunos requisitos no funcionales son:

Usabilidad
Interfaz fácil e intuitiva de utilizar por el usuario.

Interoperabilidad
Integración con sistemas externos, en este caso, se ha integrado una base de datos.

Portabilidad
El sistema es compatible con diferentes plataformas y entornos de ejecución de Java.

Escalabilidad
El sistema mantiene el rendimiento y la calidad, aunque aumenten las demandas de los usuarios en el mismo.

Disponibilidad
El sistema se mantiene operativo siempre exceptuando cuando se dan casos de mantenimiento programado y emergencias.

Seguridad
El sistema mantiene de forma segura los datos de los usuarios por lo que no es posible ser modificados o perdidos.

3. Plan de gestión de configuración

3.1 Objetivos:

Los objetivos de la gestión de configuración de nuestro grupo de trabajo han sido los siguientes:

1. Garantizar la integridad del código fuente y de los diferentes recursos del proyecto.
2. Facilitar la colaboración de los distintos integrantes del equipo.
3. Controlar y registrar cambios en el código fuente y en la configuración del proyecto.
4. Garantizar la trazabilidad de los elementos de configuración.

3.2 Identificación de la configuración:

3.2.1 Código fuente

- Paquetes y clases Java.

3.2.2 Dependencias externas

- Bibliotecas y dependencias de terceros (Junit, Surefire, Spring, etc).
- Versiones específicas de JDK y otras herramientas.

3.2.3 Documentación

- Diagramas de clase y de diseño.

3.3 Control de versiones

3.3.1 Sistema de control de versiones

- Uso de un sistema de control de versiones como Git .
- Uso de un repositorio (GitHub) para subir los cambios del proyecto.

3.3.2 Estructura de ramas

- Rama principal o main
- Rama dev de desarrollo previo al pase a producción
- Creación de rama de desarrollo para las distintas funcionalidades

3.4 Gestión de cambios

3.4.1 Procedimiento de solicitud de cambios

- Utilizar solicitudes de extracción o pull requests para cambios significativos.
- Revisiones de código por otros miembros del equipo.

3.5 Construcción y despliegue

3.5.1 Automatización de Construcción:

- Configurar herramientas de construcción como por ejemplo Maven.

3.5.2 Entornos de despliegue

- Utilizar herramientas de integración continua para despliegue automatizado.

3.6 Auditoría y trazabilidad0

3.6.1 Auditorías regulares

- Mantener auditorías regulares para garantizar la consistencia entre la configuración y la implementación.

3.7 Respaldo y restauración

3.7.1 Procedimientos de restauración

- Documentar procedimientos para restaurar el entorno de desarrollo en caso de fallos.

3.8 Documentación

3.8.1 Documentación de la configuración

- Mantener una documentación actualizada de la configuración del proyecto.

3.8.2 Documentación de versiones

- Registrar cambios importantes y nuevas características en documentos de versión.

4. Control de versiones

4.1 GitHub

Se ha utilizado GitHub para el control de versiones. Las ramas utilizadas han sido las siguientes:

4.1.1 main

Es la rama principal. En ella, se realizará una fusión desde dev cuando se entregue una versión o una versión final al cliente.

4.1.2 Dev

Es la rama donde se han integrado las distintas relaciones entre interfaces y bases de datos después de haber realizado un merge en ambas.

4.1.3 funcionalidad_RegistrarUsuario

Rama en la cual se ha desarrollado la funcionalidad de registrar a un usuario en la base de datos de la biblioteca.

4.1.4 funcionalidad_RealizarReserva

Rama en la cual se ha desarrollado la funcionalidad en la que el usuario realiza una reserva de un ejemplar, lo que le permite poder disponer de él una vez devuelto el ejemplar por el usuario al que pertenece en el momento de realizar la reserva.

4.1.5 funcionalidad_aplicarPenalización

Rama en la cual se ha desarrollado la funcionalidad de aplicar una penalización a un usuario que no ha devuelto el ejemplar en el plazo de tiempo establecido.

4.1.6 funcionalidad_comprobarPenalizacion

Rama en la cual se ha desarrollado la funcionalidad de comprobar si un usuario registrado en la base de datos tiene alguna penalización, tambien puede ser que haya alcanzado un cupo máximo de libros (5) lo cual le impediría disponer de un prestamo de otro ejemplar más.

4.1.7 funcionalidad_DevoluciónEjemplar:
En esta rama se desarrolla la funcionalidad de devolución de ejemplares prestados a los usuarios a la biblioteca.

4.1.8 funcionalidad_PréstamoEjemplar
Rama en la cual se ha desarrollado la funcionalidad de préstamo de un ejemplar por parte de un administrador de una biblioteca a un usuario en concreto.

4.1.9 funcionalidad_BorrarTitulo
Rama en la que se ha desarrollado la funcionalidad de borrar título teniendo en cuenta todos sus ejemplares y relaciones con tabla autor, prestamos de ejemplares, reservas, etc.

4.1.10 funcionalidad_AñadirTitulo
Rama en la que se ha desarrollado la funcionalidad de añadir título con una funcionalidad específica para los autores para poder seleccionarlos o añadir unos nuevos.

5. Organización del proyecto

Para lograr una perfecta organización del proyecto hemos utilizado Asana que es una herramienta web que nos ha permitido crear, gestionar y colaborar en el proyecto de una forma ordenada. Esta herramienta se adapta a una metodología ágil o Scrum que es un marco de trabajo ágil basado en iteraciones cortas y regulares denominadas “sprints”, donde se entregan incrementos de valor al cliente.

Respecto a la utilización de la herramienta, todos los miembros del equipo hemos tenido asignadas distintas responsabilidades las cuales teníamos que tener terminadas en una fecha en concreto. Cada funcionalidad puede tener un responsable que será el encargado de supervisar su correcta terminación y que todos los miembros del equipo entreguen su parte del trabajo en el momento indicado.

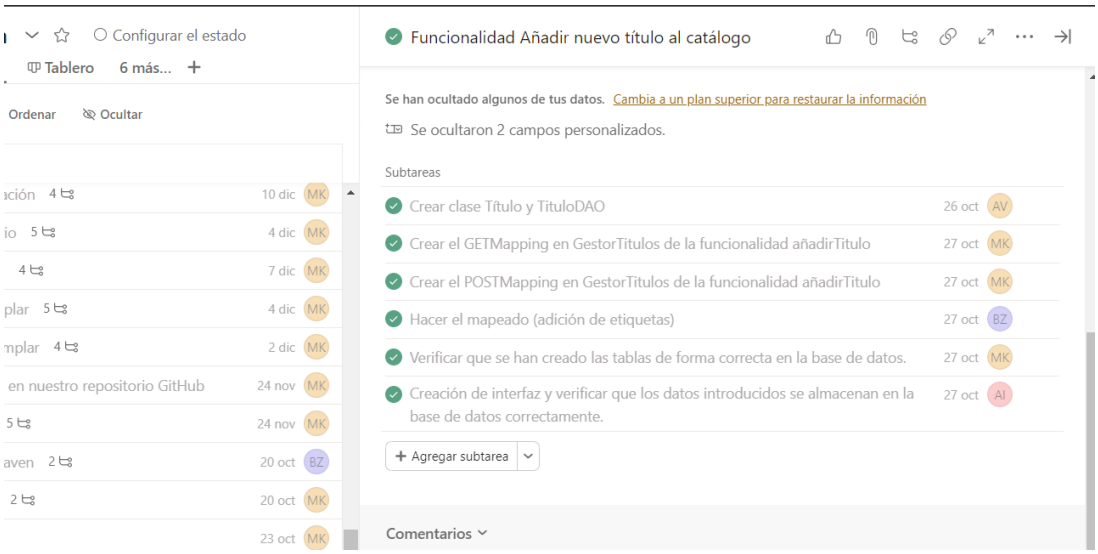
Hemos dividido la organización del trabajo en pila del sprint donde ponemos todas las nuevas tareas que se deben realizar para la correcta finalización del trabajo. A continuación, dependiendo de la prioridad planificábamos los sprints tras la finalización del anterior sprint y elegimos de la pila del sprint las tareas más urgentes de realizar para ese nuevo sprint (de una duración cada 1 de 1 semana). Creamos el sprint y lo ponemos como “En curso”, a continuación, ponemos las tareas de la pila del sprint en la pila del producto y procedemos a su desarrollo, una vez terminada una de las tareas el responsable de su finalización marca el tick y la arrastra a “finalizadas” o “en revisión” dependiendo de si la tarea necesita revisión antes de darla por finalizada.

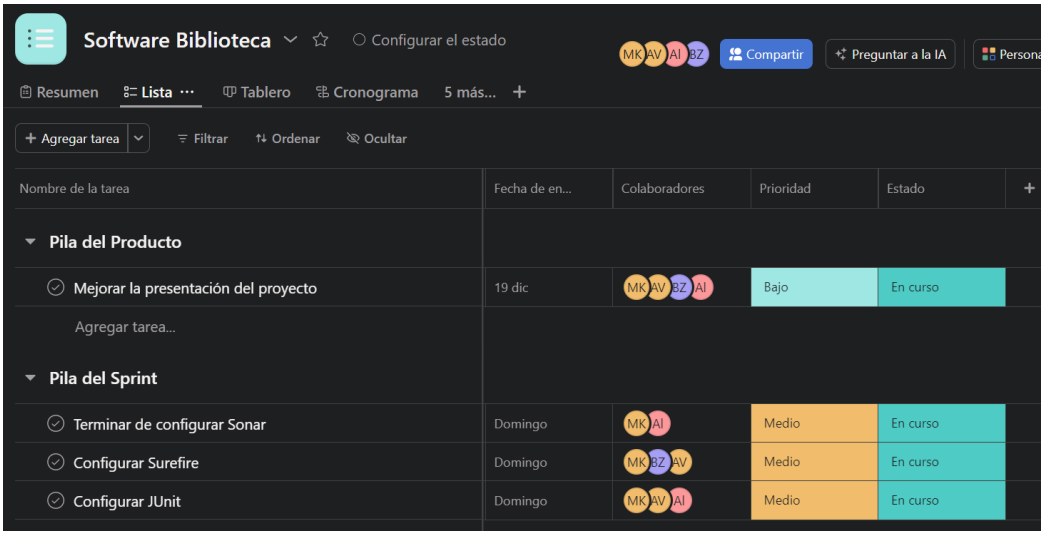
Una vez finalizadas todas las tareas de la pila del producto, se pone el sprint X en revisión y se hace una reunión de revisión en el equipo para valorar el trabajo realizado, una vez todos

conformes se da por finalizado el sprint y se procede a planificar el siguiente de sprint, de duración como hemos dicho de 1 semana.

La herramienta de Asana nos permite asignarle una prioridad y un estado a cada tarea disponible, por lo que en líneas generales la planificación de los sprints debía seguir esa norma, las tareas marcadas con la prioridad alta iban primero, luego las de medio y finalmente las de bajo, dejando estas últimas hasta el final, al no ser críticas para la adecuada consecución del proyecto.

Las tareas grandes a su vez tenían subtareas y cada 1 de esas subtareas era responsabilidad de un miembro del equipo que tenía una fecha límite para su realización:



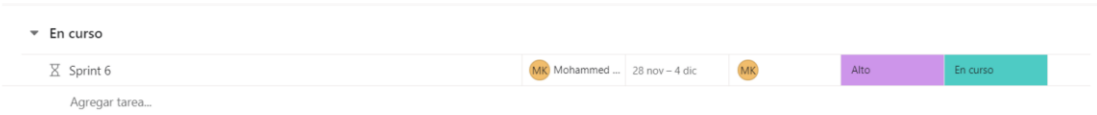


6. Sprints

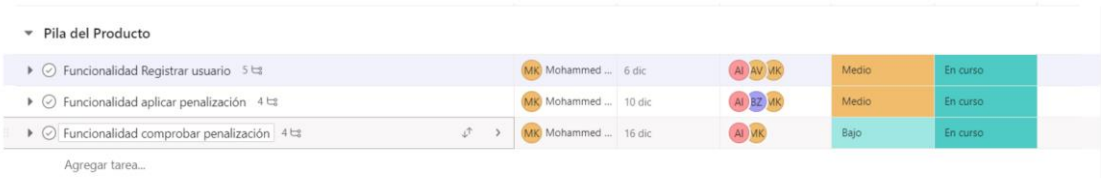
SPRINT 6(28/11/2023 – 04/12/2023):

- **Pila del Sprint**

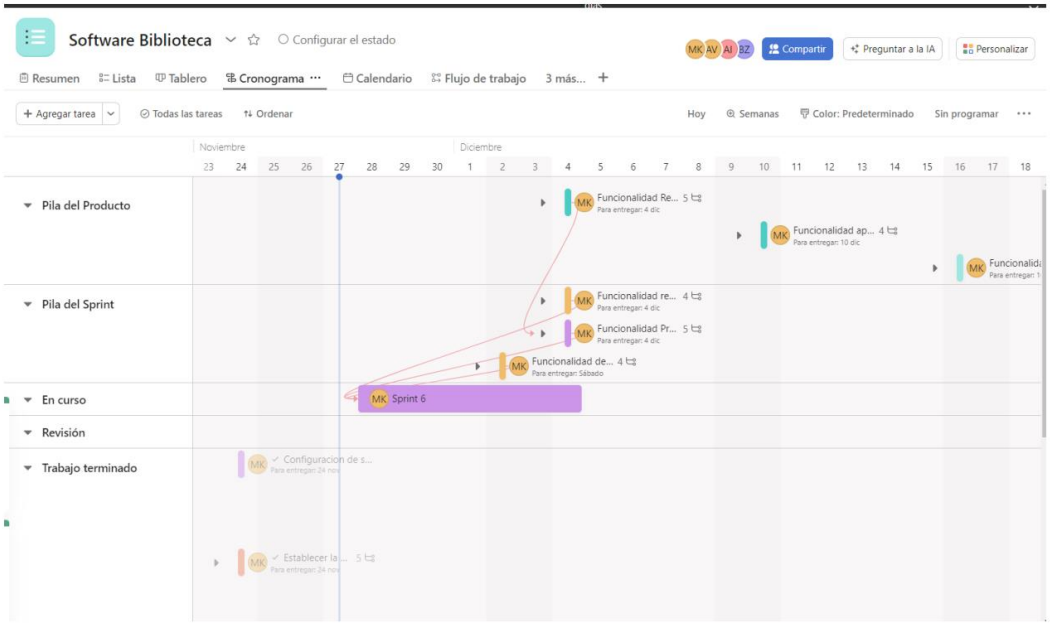
- **VISUALIZACIÓN DE LA PILA Y NOMBRE DEL SPRINT:**



- **PILA DEL PRODUCTO PENDIENTE:**



- **CRONOGRAMA DEL PROYECTO INCLUYENDO SPRINTS Y TAREAS**

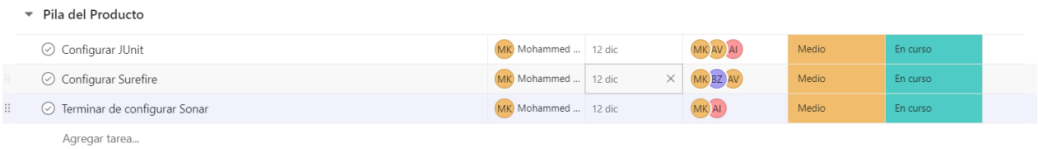


SPRINT 7(05/12/2023 – 11/12/2023):

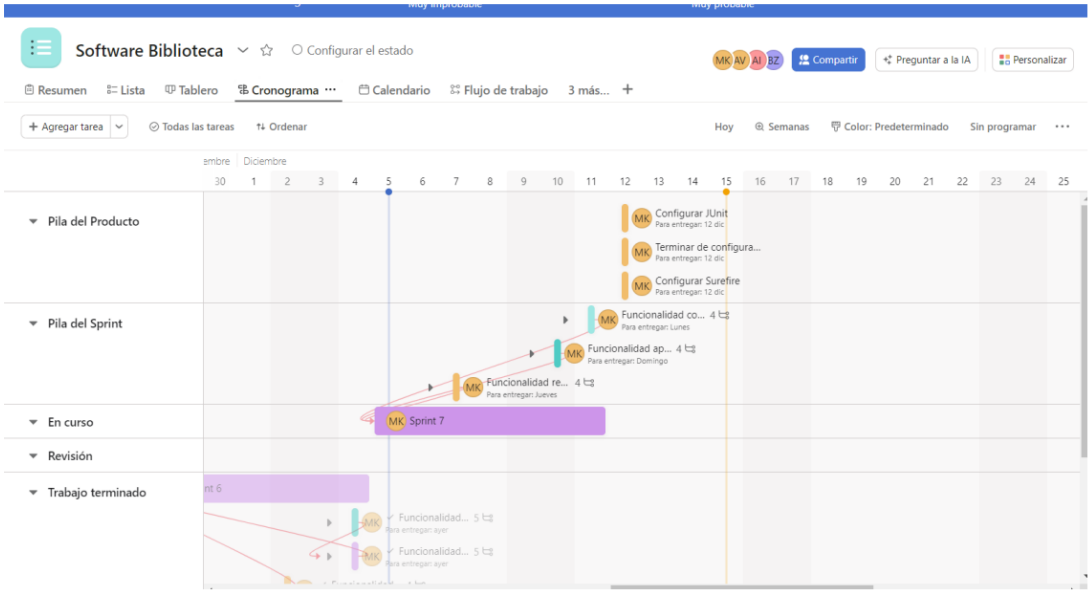
- **PILA DEL SPRINT**

- **VISUALIZACIÓN DE LA PILA Y NOMBRE DEL SPRINT:**

- **PILA DEL PRODUCTO PENDIENTE:**



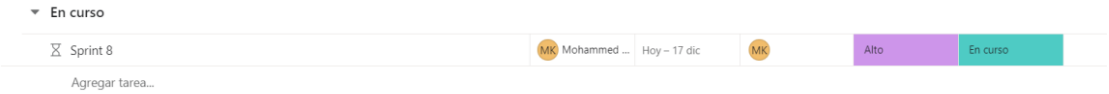
- **CRONOGRAMA DEL PROYECTO INCLUYENDO SPRINTS Y TAREAS:**



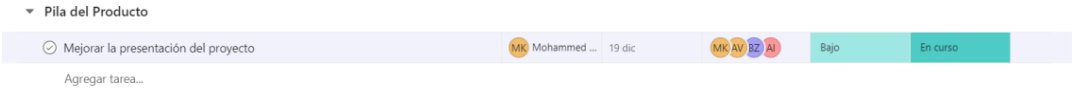
SPRINT 8(11/12/2023 – 17/12/2023):

- **PILA DEL SPRINT:**

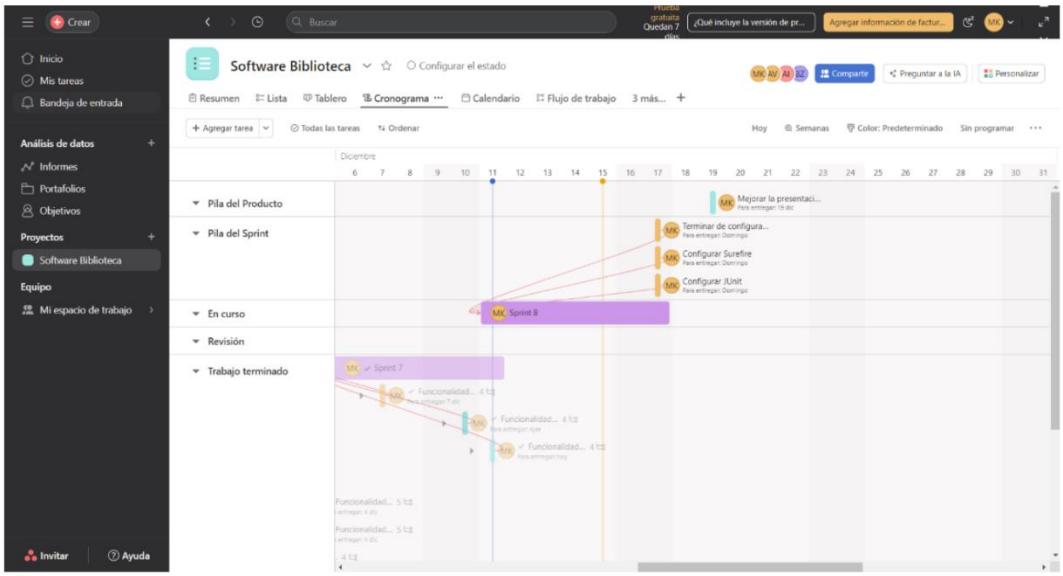
- **VISUALIZACIÓN DE LA PILA Y NOMBRE DEL SPRINT:**



- **PILA DEL PRODUCTO PENDIENTE:**



- **CRONOGRAMA DEL PROYECTO INCLUYENDO SPRINT Y TAREAS**



7. Calidad

7.1 Calidad de producto

La calidad de un producto software es el grado de en qué el software es fiel a los requisitos funcionales y no funcionales, así como con las expectativas y necesidades de los usuarios finales. El producto además está sujeto a restricciones, condiciones y revisiones, que son gestionadas tanto a nivel de usuario como a nivel de desarrollador. Este además no puede adaptarse solo a los requisitos del cliente final, sino que además debe adaptarse a las dinámicas del equipo de desarrolladores y del proyecto frente a situaciones no esperables y fracasos. Por otro lado, los estándares de calidad no son autónomos, sino que están interrelacionados con otros aspectos como el desarrollo, las pruebas y el mantenimiento ya que cada una de ellas se ve afectada por la calidad del software.

7.2 Configuración de SonarCloud

Para determinar la calidad del proyecto se ha hecho uso de SonarCloud.

Lo que hemos tenido que hacer ha sido crear una organización para permitir que SonarCloud analice el código de todos los commits. Según las diapositivas de la cuarta práctica de laboratorio, hemos añadido las líneas de código que se indican a nuestro pom.xml.

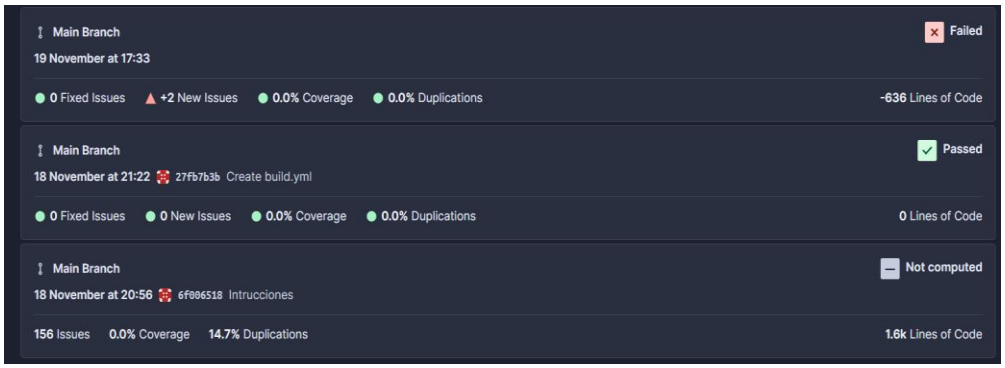
```
<properties>
    <java.version>17</java.version>
    <sonar.projectKey>mbk5574_Grupo1_ISOII</sonar.projectKey>
    <sonar.organization>mbk5574</sonar.organization>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
    <sonar.login>208144ecf93178e94ad65b40e4fb5d40a8c98cb5</sonar.login>
</properties>
```

Además, podemos asegurarnos de que Sonar está funcionando, introduciendo en la terminal el comando mvn verify sonar:sonar

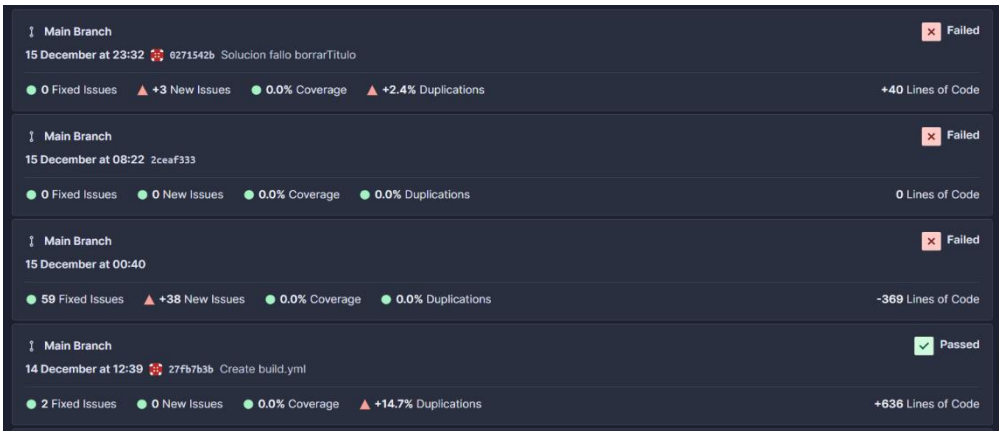
```
[INFO] Analysis report compressed in 375ms, zip size=115 KB
[INFO] Analysis report uploaded in 1876ms
[INFO] ANALYSIS SUCCESSFUL, you can find the results at: https://sonarcloud.io/dashboard?id=m
k5574_Grupo1_ISOII
[INFO] Note that you will be able to access the updated dashboard once the server has process
d the submitted analysis report
[INFO] More about the report processing at https://sonarcloud.io/api/ce/task?id=AYxz0WknNSYX-
BF-AL0
[INFO] Sensor cache published successfully
[INFO] Analysis total time: 1:21.906 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:37 min
[INFO] Finished at: 2023-12-16T18:09:25+01:00
[INFO] ------------------------------------------------------------------------
```

7.3 Análisis continuo de Sonar

El uso continuado de SonarCloud nos ha permitido detectar y corregir bugs, así como corregir vulnerabilidades del código antes de que se conviertan en problemas mayores. Además de esto, nos ha sido posible mejorar la mantenibilidad, confiabilidad y seguridad del código, de acuerdo con los estándares de calidad y las buenas prácticas de la programación. Y también, nos ha proporcionado métricas e indicadores sobre el estado actual del código, como por ejemplo la complejidad o el porcentaje de código duplicado.

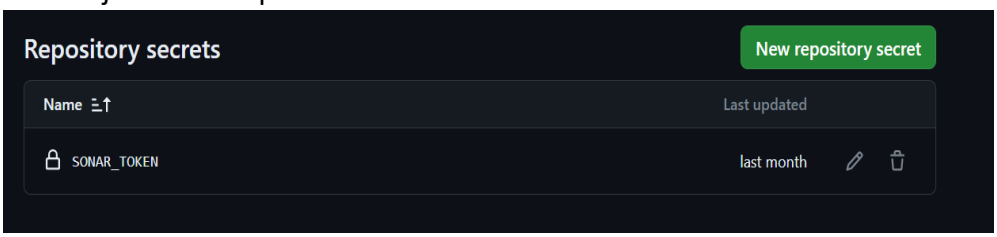


7.4 Despliegue continuo



Durante el desarrollo todo el grupo hemos contemplado la opción de integrar el análisis de SonarCloud en cada commit. Esto nos ha facilitado el análisis de nuestro código con creces de cara a mejorarlo. Para ello, hemos seguido los siguientes pasos:

1. En los ajustes del repositorio



2. Se ha añadido la siguiente dependencia al pom.xml

```
<properties>
    <java.version>17</java.version>
    <sonar.projectKey>mbk5574_Grupo1_ISOII</sonar.projectKey>
    <sonar.organization>mbk5574</sonar.organization>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
    <sonar.login>208144ecf93178e94ad65b40e4fb5d40a8c98cb5</sonar.login>
</properties>
```

3. Por último, se ha añadido el archivo de configuración básica de SonarCloud .github/workflows/build.yml para hacer el análisis después de cada push/commit.

```yaml
name: SonarCloud
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build and analyze
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
        with:
          fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: 17
          distribution: 'zulu' # Alternative distribution options are available.
      - name: Cache SonarCloud packages
        uses: actions/cache@v3
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v3
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get PR information, if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=mbk5574_Grupo1_ISOII
```

8. Testing

El testing en un proyecto es el proceso de validar y verificar que el producto o servicio que está siendo desarrollado cumple con las expectativas y requisitos del cliente. El testing es una parte esencial en un proyecto, ya que permite detectar y corregir errores, mejorar la calidad y la usabilidad y asegurar que los usuarios estén satisfechos.

En el proyecto se ha hecho uso de JUnit (Junit5) para poder ejecutar clases de forma controlada, evaluando en todo momento el funcionamiento de los diferentes métodos. Para poder utilizar Junit ha sido necesario añadir la siguiente dependencia en el pom.xml:

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

JUnit es un framework de pruebas unitarias para Java cuyo propósito principal es permitir a los desarrolladores escribir y ejecutar pruebas unitarias de forma sencilla y automatizada.

Algunas características que el grupo destaca de JUnit son:

- Asegura la calidad del código ya que nos ha permitido definir casos de prueba para garantizar que cada unidad de código funcione correctamente.
- Facilita la automatización de pruebas porque estas pueden ejecutarse repetidamente y de forma consistente.

- Facilita la identificación de errores gracias a los informes detallados que indican cuales son las pruebas que han fallado.

Por otro lado, para el testing se ha hecho uso de un marco de prueba de mocking para Java que se ha utilizado junto a JUnit. Mockito se ha utilizado para facilitar la creación y uso de objetos simulados (mocks) en pruebas unitarias.

Algunos aspectos a destacar de Mockito por el grupo son los siguientes:

- Facilita la creación de objetos simulados o mocks con el fin de simular el comportamiento de objetos reales en una prueba.
- Proporciona una sintaxis lo suficientemente intuitiva como para definir comportamientos de objetos simulados y realizar verificaciones.
- Facilita la eliminación de dependencias externas al simular el comportamiento de componentes externos, lo que ha hecho que las prueban sean más controladas y aisladas.

También queremos destacar que hemos buscado maximizar la eficacia de las pruebas mientras minimizamos el tiempo invertido y el número de casos de prueba (función mini-max).

Los objetivos que hemos buscado en todo momento han sido:

- Encontrar defectos significativos en el software, lo que es lo mismo, una mayor eficacia de las pruebas. Para ello, se han diseñado casos de prueba que sean más propensos a descubrir defectos importantes. Hemos hecho pruebas de testing principalmente en los gestores, ya que se encargan de manipular y acceder a la información de una base de datos.
- Detectar el mayor número de defectos en el software mediante pruebas efectivas. Para ello, se han diseñado casos de prueba que cumplían una amplia gama de escenarios y condiciones, con el objetivo de descubrir diferentes defectos en el sistema.
- Invertir el menor tiempo en las pruebas diseñando un conjunto mínimo pero efectivo de casos de prueba. En vez de tener una gran cantidad de casos de prueba, simplemente se han seleccionado aquellos casos que representaban los casos que pensábamos que eran más propensos a tener defectos.

8.1 Ejemplos de testing

A continuación, se van a mostrar algunos de los errores detectados durante las pruebas de testing.

A la hora de dar el alta un préstamo se permitía poner la fecha de inicio más grande que la de final, esto no debería de ocurrir. En las diferentes pruebas de testing del metodo *realizarPrestamo,* probando este caso se esperaba un error, pero en su lugar se recibió que el ejemplar se estaba guardando exitosamente. Por ende, se añadió una condición, para que en el caso de que se produjera esta situación redirigiría al usuario a la ventana de error.

```
try {
    if (prestamo.getFechaFin().before(prestamo.getFechaInicio())) {
        log.error(msg:"La fecha de fin no puede ser anterior a la fecha de inicio.");
        redirectAttributes.addFlashAttribute(attributeName:"error", attributeValue:"La
        return "redirect:/rutaErrorPrestamo";
    }
```

En la elaboración de los casos de prueba de *bajaEejemplar*, se planteó el caso en el cual no el método no recibía el id del ejemplar a borrar, esta situación no tendría que darse, pero no se puso que era obligatorio seleccionar un id, asique en el html de *bajaEjemplar* se hizo que el campo fuera obligatorio con el atributo *required*.

```
<p>Seleccione el id del Título para el ejemplar a dar de baja:
    <select name="idsEjemplares" required>
```

Pruebas realizadas:

GestorTitulos:

Se realizará casos de prueba para los siguientes métodos: *realizarPrestamo*, *realizarDevolucion y reservarejemplar.*

realizarPrestamo:

Para lograr una cobertura de uso se ha optado por definir un préstamo con todos los atributos, porque la aplicación te obliga a rellenarlos, e ir cambiando las fechas para abarcar los dos posibles casos, el primero que la fecha inicio sea menor a la fecha final y el segundo el caso opuesto, además se ha probado a lanzar una *Exception* en caso de que el método *prestamoService.realizarPrestamo* lanzara un error.

realizarDevolucion:

En este caso se plantea dar los siguientes valores de pruebas: un id que no existe, un préstamo con una fecha final de entrega superior al día en el que se devuelve y su caso contrario. Con estos valores se conseguiría una cobertura de caso de uso.

reservarEjemplar:

Los diferentes parámetros de prueba que se han utilizado para lograr la cobertura de uso son los siguientes: un ejemplar con el parámetro disponible igual a true, otro con disponible false, un ejemplar con id que no se encuentra en la base de datos y un ejemplar que si este en la base datos pero que no tenga el atributo título es decir sea nulo.

Con estas pruebas realizadas se conseguiría una cobertura condición y decisión para el boque *GestorTitulos*.

GestorPrestamos:

Se comprobará con diferentes casos de prueba los siguientes métodos: *altaTitulo*, *actualizarTitulo*, *borrarTitulo, altaEjemplar* y *bajaEjemplar.*

altaTitulo:

Se usarán los siguientes valores para las diferentes pruebas: una lista con dos autores con todos los parámetros bien definidos para que no salten excepciones, y otra donde el tipo de libro no sea libro o revista.

actualizarTitulo:

Para una prueba se definirán dos títulos uno representará el de la base de datos y otro tendrá el mismo *id* pero diferentes valores en los atributos para simular la actualización. Otra prueba consistirá en definir un título y que este no esté en la base de datos.

borrarTitulo:

Tendrá dos pruebas una para su correcto funcionamiento, otra el cual simula que no se encuentre el isbn del título a borrar. Para ello la primera prueba tendrá como parámetros dos títulos existentes uno que se podrá borrar y otro que no, la segunda un isbn que no se encuentre en la base de datos y la segunda un isbn con valor *null*.

altaEjemplar:

Se definirán dos pruebas igual que la anterior una será para su comprobar su correcto funcionamiento y otra para cuando no se encuentre el id del título. Se usará en el primer caso un título y un ejemplar que herede del mismo título, y en el segundo caso se definirá un título inexistente.

bajaEjemplar:

Este método tendrá tres casos de prueba uno para comprobar que se de baja a los ejemplares, otro para ver en caso de que fallara la eliminación y por último uno que reciba una lista vacía. En el primer caso se definirá una lista con ides correspondientes a ejemplares que estén en la base de datos, el segundo con una lista de ides asociados a ejemplares disponibles y el tercero tendrá una lista vacía.

GestorUsuario:

De esta clase se probará el correcto funcionamiento de registrar el usuario, mostrar tanto el formulario de registro como el de resultado.

mostrarFormularioRegistroUsuario y mostrarResultadoRegistro :

Simplemente se comprueba si el enlace que envía es el correcto.

registrarUsuario:

Tendrá dos pruebas una con su correcto funcionamiento y otra donde se lanza una *exception*. Para ambas se define un usuario nuevo.

PenalizacionService:

Tiene las siguientes funciones aplicarPenalizcion y comprobarPenalizacion.

aplicarPenalizacion:

Se realizarán las diferentes pruebas con uno de estos parámetros: un usuario con la fecha final del préstamo pasada, otro con la fecha final presente y por último uno con una fecha final futura.

comprobarPenalizacion:

Con los siguientes valores se crearán diferentes las pruebas: un usuario con la fecha final del préstamo pasada, otro con la fecha final presente, uno con una fecha final futura y por último un usuario nulo.

Con estos valores se implementarán diferentes casos de prueba para lograr la cobertura de condición y decisión.

PrestamoService:

Tiene dos métodos *puedePrestar* y realizarPrestamo.

puedePrestar:

Como parámetros de prueba se usarán: un usuario que haya alcanzado el límite de préstamos, otro con penalizaciones y uno sin los anteriores parámetros.

realizarPrestamo:

Se realizarán dos casos de prueba, uno con un usuario con penalización, y otro con un usuario sin penalización

TituloService:

Tiene implementada los siguientes metodos: *eleminarTitulosConVerificaciones, eliminarTitulosYAutores, eliminarEjemplarConVerificaiones, obtenerAutores y guardarTitulo.*

elminarTitulosConVerificaciones:

Sus valores de prueba serán: título existente, titulo no existente, un ejemplar que no esté disponible.

eliminarTitulosYAutores:

Tendrá como valores de prueba: títulos con autores, un título que sea libro, un título que sea revista y un título que no sea ninguna de estas dos anteriores opciones.

eliminarConVerificaciones:

Los valores de prueba serán: ejemplar que no existe, ejemplar no disponible, ejemplar con préstamo activo, ejemplar con reserva y un ejemplar con los anteriores valores negados.

obtenerAutor:

Usará como valores de prueba diferentes listas con nombres de autores: una con solo nombres existentes, otra con nombres de autores no existentes y una lista con nombres autores existentes como no existentes.

guardarTitulo:

Tendrá dos valores de prueba, el primero es un título cunado es revista y el segundo cuando un título tiene un tipo no valido.

9. Mantenimiento

La estrategia de mantenimiento del proyecto ha consistido en usar SonarCloud con el fin de identificar y resolver los problemas en el código. Cada vez que se solucionan esos problemas, se vuelve a realizar un nuevo análisis de calidad.

Aparte de solucionar estos problemas hemos seguido los principios de clean code, basado en los principios SOLID.

- **Single Responsibility Principle:** una clase solo debería tener una razón para el cambio. Por ejemplo la clase ejemplar, es una clase para todos los ejemplares, la clase Autor es una clase para todos los autores.

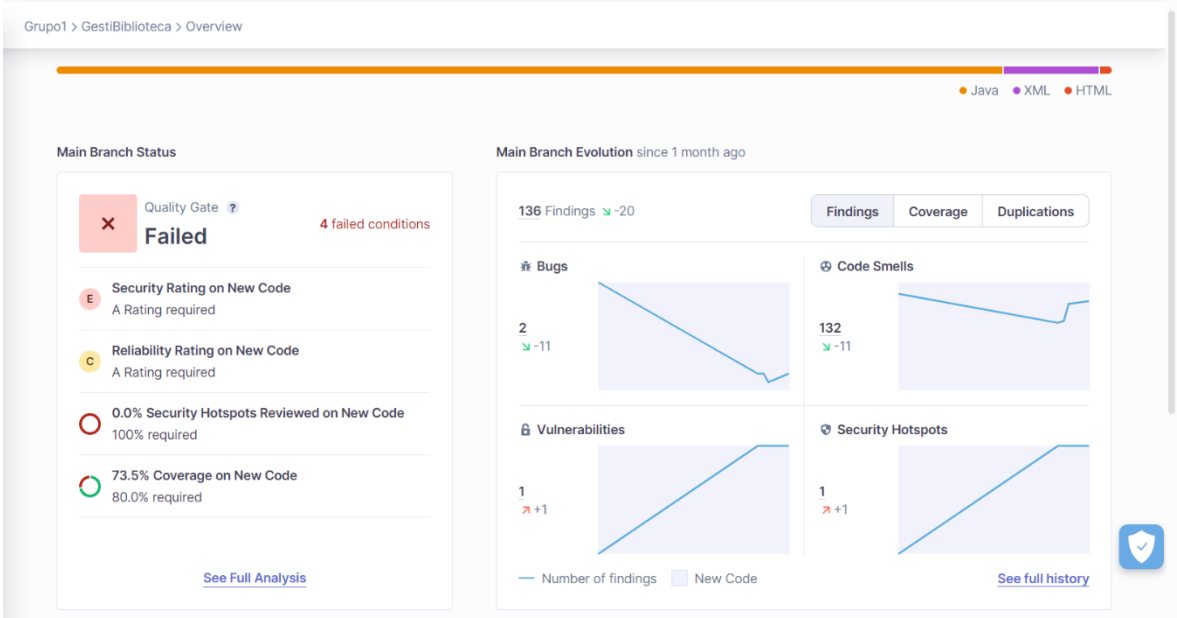
- **Open Closed Principle:** para cumplir este principio lo que se ha hecho ha sido dejar abiertas las clases a posibles extensiones y no a modificaciones. Por ejemplo, la clase Libro no se puede modificar pero sí extender, usando una herencia para formar nuevos tipos de libros.

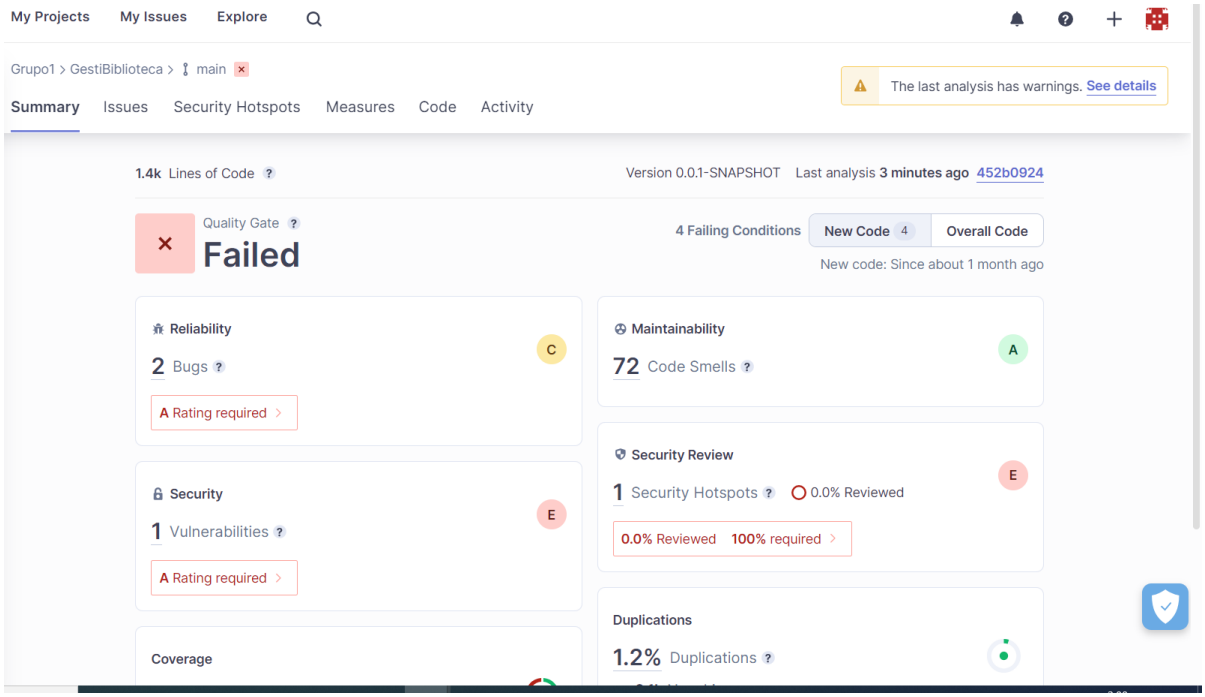
- **Liskov's Substition:** cada clase hereda de otra que puede usarse como su padre sin ser necesario tener conocimiento de las diferencias entre ellas.

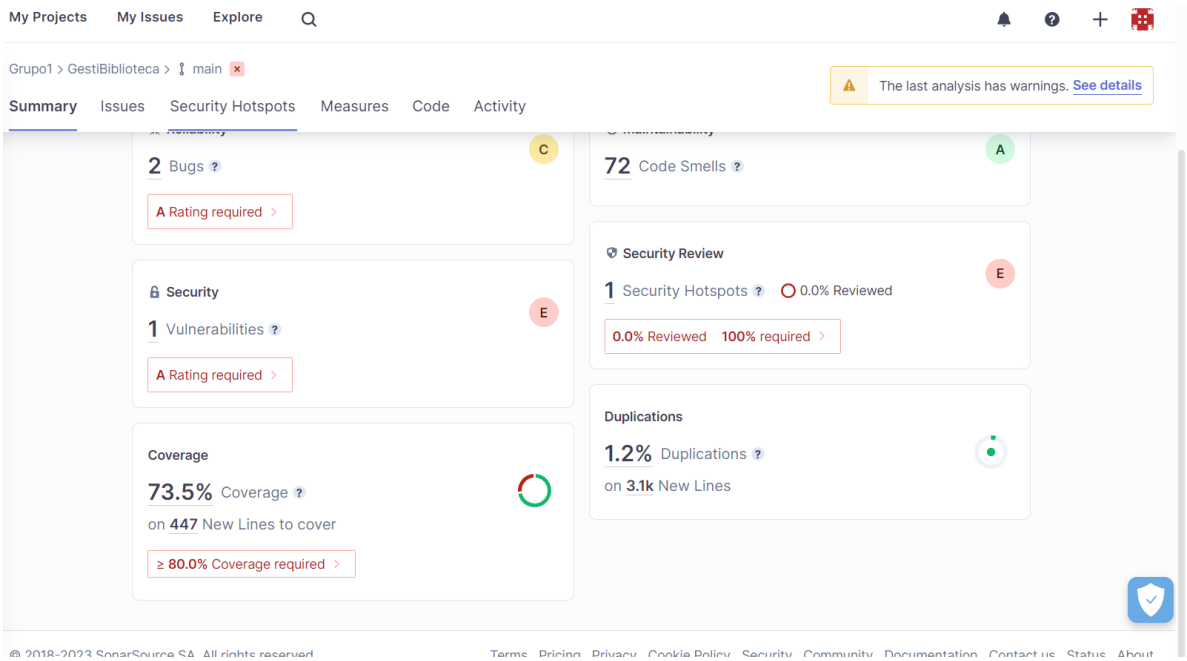
- **Interface Segregation Principle:** las clases implementadas, no implementan métodos que innecesarios, todos los métodos de las clases son necesarios.

- **Dependency Inversión Principle:** una clase no tiene que depender directamente de otra clase para funcionar, pero sí de una interfaz.

Empezamos con el mantenimiento de nuestro proyecto, hacemos un mvn verify sonar:sonar para ejecutar un analisis en sonarcloud de nuestro proyecto, obtenemos los siguientes resultados:

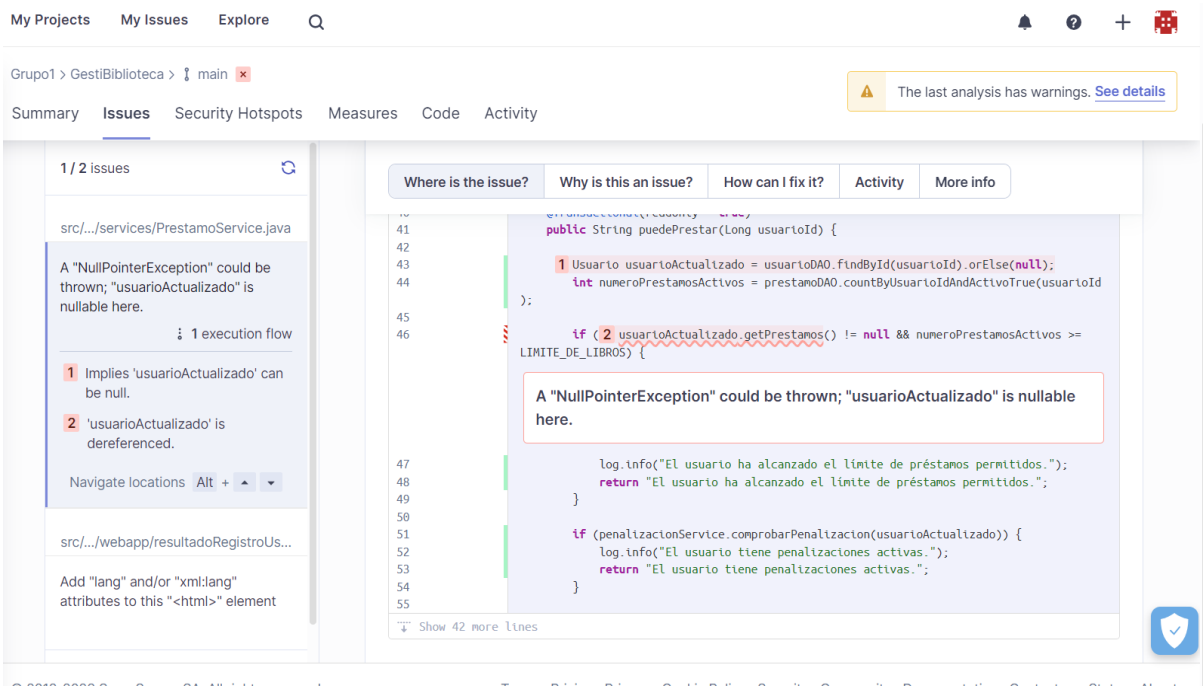




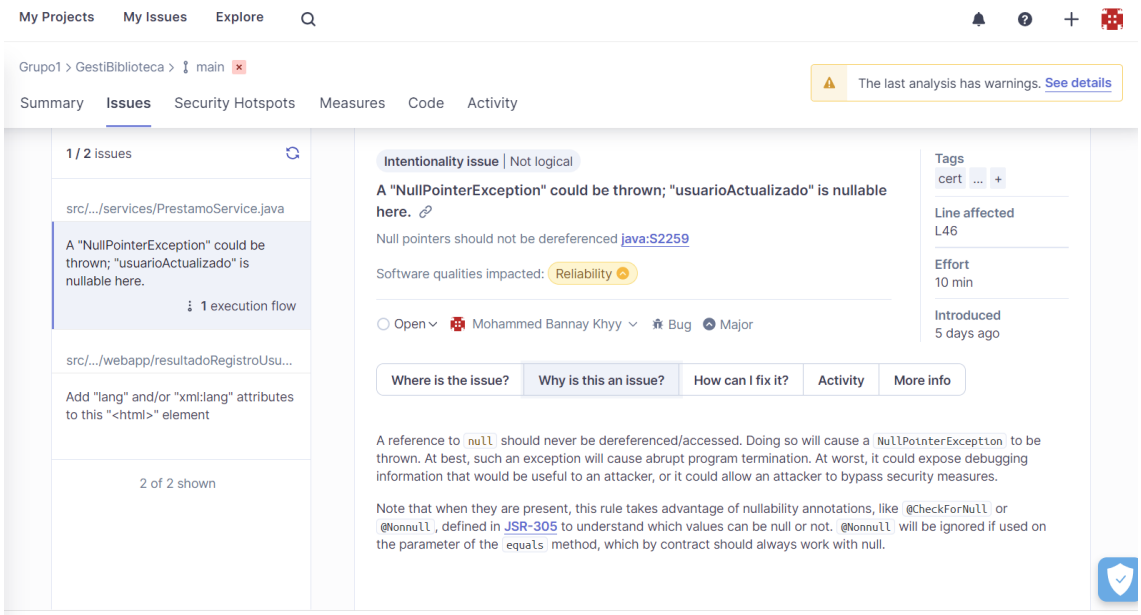


Tras analizar los errores que sonarcloud indica en nuestro proyecto, llegamos a la conclusión de que los errores más urgentes de solucionar en nuestro proyecto son los categorizados como Bugs. Por lo tanto, nuestro mantenimiento irá centrado principalmente en la solución de estos dejando los code smells, duplicaciones, vulnerabilidades para futuros mantenimientos de la aplicación.

Empezamos con el primer bug:



La razon por la que esto es un bug también nos lo indica sonarcloud:

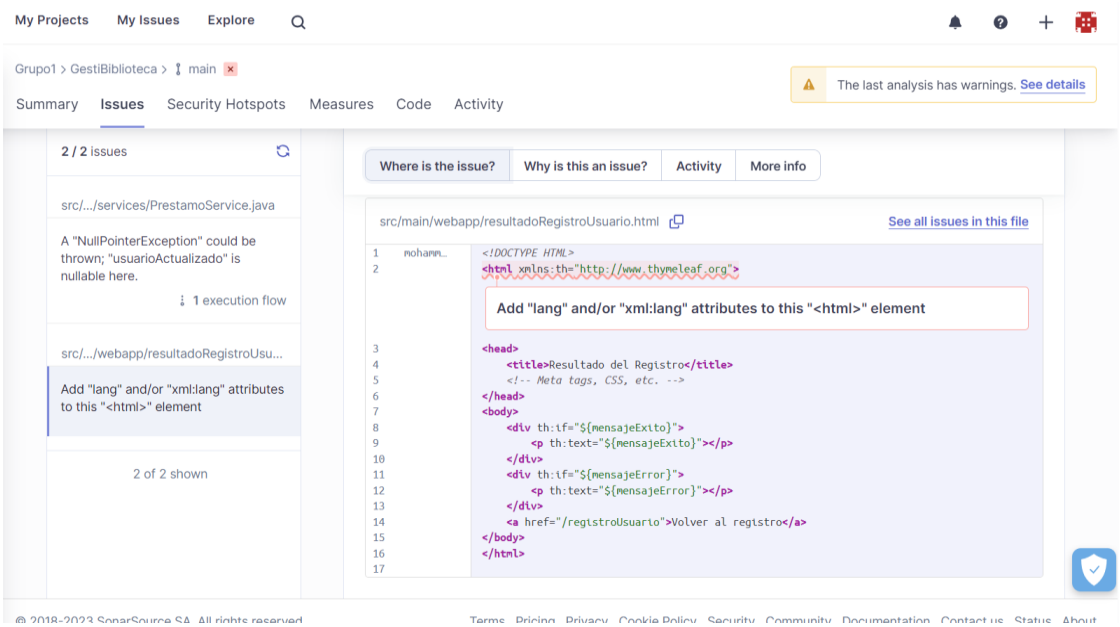


Esta advertencia de SonarQube se refiere a un problema común en la programación, especialmente en lenguajes como Java, donde el acceso a una referencia nula puede causar problemas significativos.

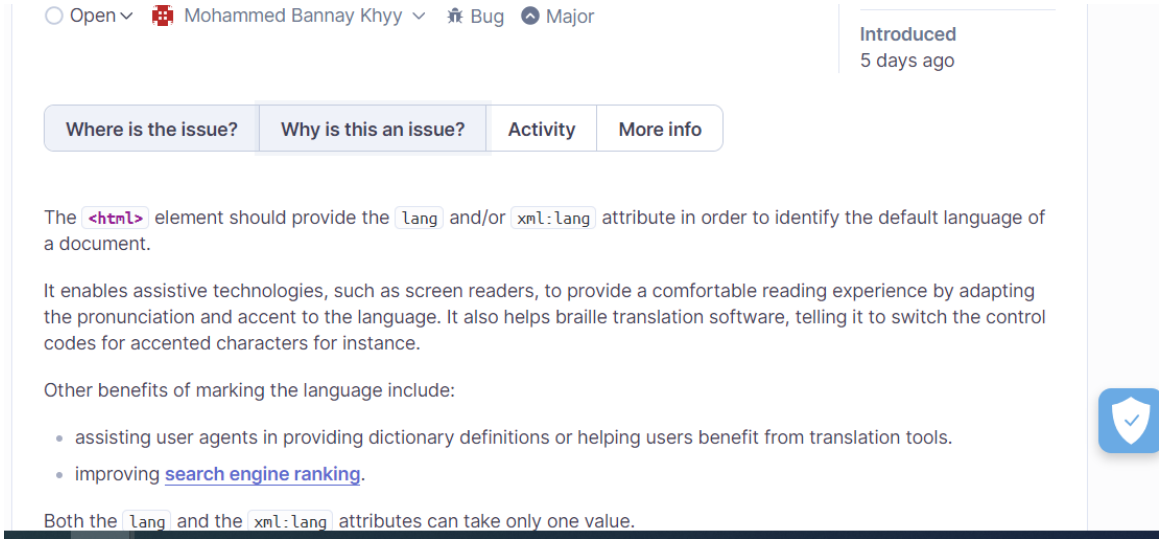
Una de las soluciones que nos da sonarcloud es que comprobemos que eso que puede dar nulo es == null en cuyo caso hacemos un return y el codigo siguiente sería inalcanzable.

Procedemos a realizar las modificaciones en nuestro código y con esto damos por solucionado el primer bug.

Seguimos con el segundo bug:

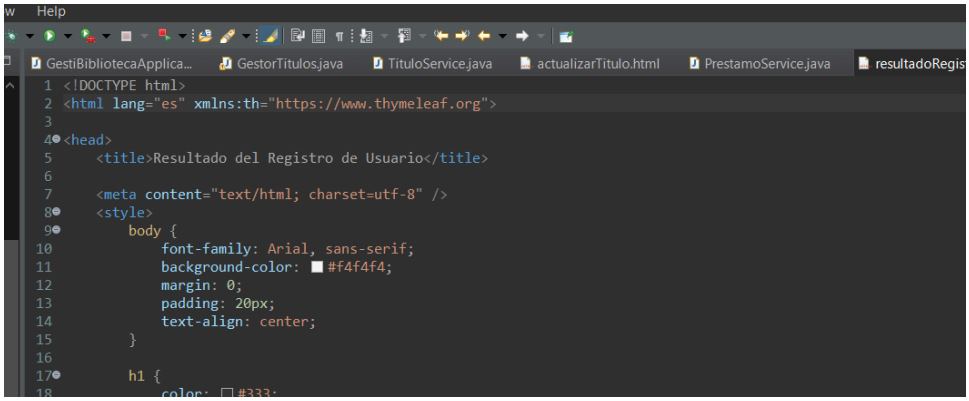


La razón por la que esto es un bug nos lo indica como antes sonarcloud:



El error marcado nos dice que la ausencia del atributo **lang** en el elemento **<html>** de un documento HTML se considera una infracción porque este atributo juega un papel crucial en varias áreas.

Solucionamos el error indicando el atributo de idioma:



Y ejecutando de nuevo el análisis todos los bugs desaparecen.