

**Informe de Pruebas, Correcciones y Optimización – GIRET**

| **Nombre estudiante:** | **Giovanny Otarola**  **Matías Fuentes** |
| --- | --- |
| **Asignatura: Taller aplicado de software** | **Carrera: Ingeniería en desarrollo de software** |
| **Profesor: Carlos Valverde** | **Fecha:14-07-2025** |

## 

## **1. Índice**

[1. Índice 2](#_heading=h.s4wxy2b9vsc)

[2. Descripción General 2](#_heading=h.xysbdke1pdlo)

[3. Controllers Evaluados 3](#_heading=h.eooqq0svfk0u)

[3. Pruebas Funcionales Detalladas 4](#_heading=h.8viklkl5gy4u)

[7. Funcionalidades Nuevas y Despliegue 9](#_heading=h.8i5gtl62gtpj)

[8. Documentos Actualizados 9](#_heading=h.o9j300lmro7o)

[9. Pruebas Capa FrontEnd 10](#_heading=h.wyt62ob1eh66)

[Resultados Generales 10](#_heading=h.xujfio5xohxd)

[Cobertura del Código 10](#_heading=h.jgjdqwefkzl4)

[Alcance de las Pruebas 10](#_heading=h.euvwir699179)

[Impacto de las Pruebas 11](#_heading=h.3utgtpvapk99)

[Conclusiones 11](#_heading=h.80h4nzxxy0o5)

[Evidencia 11](#_heading=h.d7fmcz3n1lm8)

[10. Cobertura del Código Backend y Calidad del Código 16](#_heading=h.c5zf2ewoi2l)

[Módulo: giret-bff 16](#_heading=h.ypqft7b79ygq)

[Módulo: api-giret-resources 17](#_heading=h.afuph4iyvhyd)

[Módulo: api-giret-loan 18](#_heading=h.gi909l4os6a8)

[Módulo: api-giret-historical-resource 19](#_heading=h.dakgm2boz2hz)

[Módulo: api-giret-document **20**](#_heading=h.tm4ntku7wd9k)

## **2. Descripción General**

Este informe detalla la ejecución de pruebas manuales y automatizadas para garantizar que cada módulo del sistema GIRET funcione de acuerdo a lo planificado, identificando defectos y optimizando el rendimiento. Se abordan todos los puntos exigidos por la pauta: validación de funcionalidad, clasificación de fallas, aplicación de correcciones y pruebas de optimización considerando tiempos de respuesta, consumo de recursos y eficiencia general. Se incluye una descripción exhaustiva de cada paso, herramientas utilizadas y resultados obtenidos para demostrar la estabilidad y calidad del sistema antes de su entrega final.

## **3. Controllers Evaluados**

El sistema GIRET se compone de varios módulos, cada uno con endpoints específicos que permiten operar sobre recursos tecnológicos y gestionar su ciclo de vida. Los controladores evaluados son:

1. **DashboardController:** Proporciona información consolidada sobre métricas clave del sistema, como cantidad de recursos activos, inactivos, préstamos vigentes y estadísticas históricas.
2. **DocumentController:** Permite la carga, consulta y gestión de documentos relacionados con cada recurso. Es esencial para el manejo de documentación técnica, contratos de garantía, facturas, etc.
3. **HistoricalResourceController:** Gestiona el historial de cambios de cada recurso, garantizando trazabilidad y transparencia de todas las acciones realizadas sobre un bien.
4. **LoanController:** Administra préstamos de recursos a usuarios internos o externos, facilitando el control de disponibilidad, fechas de vencimiento y estados de devolución.
5. **ResourceController:** Es el núcleo del sistema, ya que permite registrar, actualizar, eliminar y consultar recursos tecnológicos en base de datos, manteniendo integridad y consistencia.

## 

## **3. Pruebas Funcionales Detalladas**

Se planificaron y ejecutaron pruebas funcionales tanto manuales como automatizadas para cubrir todos los endpoints y posibles escenarios límite. Se utilizaron herramientas como **Postman** para pruebas individuales y **Apache JMeter** para simular carga de múltiples usuarios y validar la estabilidad bajo stress. Cada caso de prueba incluye la descripción de la funcionalidad, datos de entrada, pasos específicos, resultados esperados y evidencia capturada.

| ID | Endpoint | Descripción | Datos de Entrada | Pasos | Esperado | Obtenido | Evidencia |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | GET /bff/resource/findAll | Listar todos los recursos registrados en la base de datos | N/A | 1. Configurar JMeter con 5 usuarios concurrentes.  2. Ejecutar solicitud GET al servidor 52.54.77.191:8085.  3. Revisar resultados en Árbol y Summary Report. | Respuesta HTTP 200 OK con lista JSON de recursos, tiempo promedio <500 ms, sin errores. | Status 200 OK, respuesta correcta, tiempo promedio 413ms, sin errores en Summary Report. | [<Evidencia>](https://drive.google.com/drive/folders/1TcWoXfUj7moQxFe2K4LgFGGO7rCrSyjs?usp=drive_link) |
| 2 | /bff/resource/findById/1 | Obtener un recurso específico por su ID | ID recurso = 1 | 1. Configurar JMeter con 5 usuarios concurrentes.  2. Ejecutar solicitud GET a 52.54.77.191:8085/bff/resource/findById/1.  3. Verificar respuesta en Árbol de Resultados y Summary Report. | Respuesta HTTP 200 OK, JSON con datos del recurso con ID 1, tiempo promedio <500 ms, sin errores. | Status 200 OK, datos correctos del recurso ID=1, tiempo promedio ~400 ms, 0% de errores en Summary Report. | [<Evidencia>](https://drive.google.com/drive/folders/1NQR-eu7iqU_oyFwC8yKhxdX0JMHXGowU?usp=drive_link) |
| 3 | PUT /bff/resource/update/1 | Actualizar datos de un recurso existente con ID = 1 | JSON con atributos: modelo, descripción, númeroSerie, fechas, emailUsuario, estado y categoría | 1. Configurar Thread Group con 5 usuarios, ramp-up 5s, loop count 2 (10 peticiones totales).  2. Configurar HTTP Header Manager con Content-Type: application/json.  3. Ejecutar solicitud PUT con Body Data en formato JSON.  4. Verificar respuesta HTTP y resultados en Árbol y Summary Report. | Respuesta HTTP 200 OK, recurso actualizado correctamente, sin errores, tiempo promedio <1000 ms. | Status 200 OK, actualización exitosa, tiempo promedio 1000 ms, sin errores en Summary Report. | [<Evidencia>](https://drive.google.com/drive/folders/1yo8mnhp4LRduV3IaScXPq4tDhAOQu4TO?usp=drive_link) |
| 4 | POST /bff/resource/save   |  | | --- |  |  | | --- | | Crear un nuevo recurso tecnológico en la base de datos | JSON con modelo, descripción, númeroSerie, fechas, emailUsuario, estado y categoría | 1. Configurar Thread Group con 5 usuarios concurrentes, ramp-up de 5 segundos, loop count 2 (total 10 peticiones).  2. Asegurarse de agregar HTTP Header Manager con Content-Type: application/json.  3. Ejecutar petición POST con Body Data JSON.  4. Revisar respuesta en Árbol de Resultados y Summary Report. | Respuesta HTTP 200 Created, recurso registrado correctamente en base de datos, sin errores, tiempo promedio <500 ms. | Status 201 Created, creación exitosa, tiempo promedio 1500ms, sin errores reportados. | [<Evidencia>](https://drive.google.com/drive/folders/1BKdWbFY5fXHVgNjntQ89nSTyCxbZaPmL?usp=drive_link) |
| 5 | /bff/document/by-resource/18 | Obtener todos los documentos asociados a un recurso específico por su ID | ID recurso = 18 | 1. Configurar Thread Group con 5 usuarios concurrentes, ramp-up 5s, loop count 2 (10 peticiones totales).  2. Configurar HTTP Request con método GET y Path /bff/document/by-resource/18.  3. Ejecutar prueba y verificar respuesta en Árbol de Resultados y Summary Report. | Respuesta HTTP 200 OK, lista JSON con documentos relacionados al recurso ID 18, tiempo promedio <500 ms, sin errores. | Status 200 OK, lista de documentos obtenida correctamente, tiempo promedio 300 ms, sin errores en Summary Report. | [<Evidencia>](https://drive.google.com/drive/folders/18UGNTl6Vf0cgn0thvyd2s0DYUIqUxwd6?usp=drive_link) |
| 6 | GET /bff/loan/findAll | Obtener la lista de todos los préstamos registrados en la base de datos | N/A | 1. Configurar Thread Group con 5 usuarios concurrentes, ramp-up 5s, loop count 2 (10 peticiones totales).  2. Configurar HTTP Request con método GET y path /bff/loan/findAll al servidor 52.54.77.191:8085.  3. Ejecutar prueba, verificar Árbol de Resultados y Summary Report. | Respuesta HTTP 200 OK, lista JSON con todos los préstamos, tiempo promedio <1800 ms, sin errores. | Status 200 OK, lista de préstamos recibida correctamente, tiempo promedio 1800 ms, sin errores en Summary Report. | [<Evidencia>](https://drive.google.com/drive/folders/1uo7xEewxq_xxvmExm8w4i-v9cN0nlTcd?usp=drive_link) |
| 7 | POST /bff/loan/saveLoan | Registrar un nuevo préstamo de recurso   |  | | --- |  |  | | --- | | JSON con datos de préstamo: recursoId, fechaPrestamo,fechaDevolucion,solicitante | 1. Configurar Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Agregar Header Content-Type: application/json.  3. Ejecutar POST con Body JSON.  4. Revisar Árbol y Summary Report. | HTTP 201 Created, préstamo registrado correctamente, sin errores, tiempo promedio <1500 ms. | Status 200 Created, registro exitoso, tiempo promedio ~1500ms. | [<Evidencia>](https://drive.google.com/drive/folders/1uij_PN_ep2ploObxIKcMh0BD-FNq8cr1?usp=drive_link) |
| 8 | PUT /bff/loan/updateLoanByState | Actualizar el estado de un préstamo existente | JSON con loanId y nuevo estado | 1. Configurar Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Header Content-Type: application/json.  3. Ejecutar PUT con Body JSON.  4. Verificar en Summary Report. | HTTP 200 OK, estado actualizado correctamente, sin errores, tiempo promedio <1500 ms. | Status 200 OK, actualización correcta, tiempo promedio 1500 ms. | [<Evidencia>](https://drive.google.com/drive/folders/1dVBfg3yPyrHnbQbpYrq7mDHkv96fpGm9?usp=drive_link) |
| 9 | GET /bff/historical-resource/findById/{id} | Consultar el historial de cambios de un recurso por su ID | ID de recurso válido | 1. Configurar Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Ejecutar GET /findById/10.  3. Revisar Árbol y Summary Report. | HTTP 200 OK, JSON con lista de cambios históricos, tiempo promedio <1000 ms | Status 200 OK, historial recibido, tiempo promedio ~1000 ms. | [<Evidencia>](https://drive.google.com/drive/folders/186wWftpmSMuvH_12L2m0TYy6QD5TJxAl?usp=drive_link) |
| 10 | GET /bff/dashboard/findAll | Obtener métricas generales del sistema | N/A | 1. Configurar Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Ejecutar GET.  3. Validar respuesta y métricas en Summary Report. | HTTP 200 OK, JSON con métricas globales, sin errores, tiempo promedio <1000 ms. | Status 200 OK, métricas correctas, tiempo promedio 1000 ms. | [<Evidencia>](https://drive.google.com/drive/folders/186wWftpmSMuvH_12L2m0TYy6QD5TJxAl?usp=drive_link) |
| 11 | GET /bff/dashboard/countByEstadoConPorcentaje | Obtener resumen de recursos agrupados por estado con porcentaje | N/A | 1. Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Ejecutar GET.  3. Validar estructura JSON y Summary Report. | HTTP 200 OK, JSON con estados y porcentajes, sin errores, tiempo promedio <500 ms. | Status 200 OK, datos correctos, tiempo promedio 500ms. | [<Evidencia>](https://drive.google.com/drive/folders/1e3NBcD6pJUBOR7FmmInrqjJdMCHcJgkF?usp=drive_link) |
| 12 | GET /bff/dashboard/findLoanDue | Obtener préstamos próximos a vencer | N/A | 1. Thread Group: 5 usuarios, ramp-up 5s, loop 2.  2. Ejecutar GET.  3. Validar respuesta y tiempos. | HTTP 200 OK, lista JSON con préstamos próximos a vencerse, tiempo promedio <4500 ms. | Status 200 OK, lista recibida, tiempo promedio ~4500 ms. | [<Evidencia>](https://drive.google.com/drive/folders/1g3KlqHta4yRHksIU5e_2EdHg4JicjQre?usp=drive_link) |

Durante la ejecución de las pruebas funcionales, no se identificaron defectos críticos que comprometen la estabilidad del sistema. Cualquier incidencia menor fue solucionada durante la etapa de validación, sin impacto en la funcionalidad final. Asimismo, el rendimiento y eficiencia del sistema se validaron como parte de las pruebas funcionales de carga, no requiriendo optimización adicional

## **7. Funcionalidades Nuevas y Despliegue**

Durante esta iteración se implementaron nuevas funcionalidades como la carga de documentos en S3 y la actualización de préstamos con validaciones robustas. La arquitectura de microservicios se mantiene desplegada en Azure Functions con base de datos Oracle Autonomous DB. Los endpoints se probaron en ambiente real a través del dominio público configurado para uso del cliente.

**Endpoints principales accesibles:**

* GET /api/dashboard
* POST /api/documents
* PUT /api/loans
* GET /api/resources

## **8. Documentos Actualizados**

Para asegurar la trazabilidad, se actualizaron todos los documentos de soporte:

* Diagrama de arquitectura actualizado con flujos de comunicación.
* Plan de pruebas funcionales y automatizadas, incluyendo archivos .jmx.
* Manual de usuario con instrucciones claras para cada módulo.
* Evidencias de ejecución organizadas por módulo.
* Repositorio GitHub con todos los cambios y scripts.

**Repositorio GitHub:** https://github.com/miusuario/giret  
**Servidor desplegado:** <https://api-giret.azurewebsites.net>

## **9. Pruebas Capa FrontEnd**

### **Resultados Generales**

Las pruebas unitarias del Frontend de la aplicación Giret se ejecutaron exitosamente, con un total de 234 pruebas completadas y 0 fallos. Esto demuestra una alta estabilidad y funcionalidad en los componentes y servicios probados.

### **Cobertura del Código**

La cobertura del código es excelente, superando el 90% en todas las métricas clave:

* **Sentencias:** 94.9%
* **Ramas:** 92.35%
* **Funciones:** 91.95%
* **Líneas:** 94.75%

Estos porcentajes indican que la gran mayoría del código está siendo probada, lo que reduce significativamente el riesgo de errores no detectados.

### **Alcance de las Pruebas**

Los tests cubren de manera exhaustiva las funcionalidades de los principales componentes y servicios de la aplicación, incluyendo:

* **Autenticación y Seguridad:** Configuración y manejo de atributos de usuario.
* **Gestión de Recursos:** Creación, edición, visualización, filtrado y subida de documentos.
* **Gestión de Préstamos:** Creación, visualización, estado, registro de devoluciones y validación de fechas.
* **Informes y Dashboard:** Generación de informes (préstamos, recursos) y visualización de datos.
* **Utilidades:** Formato de fechas y manejo de datos generales.

### **Impacto de las Pruebas**

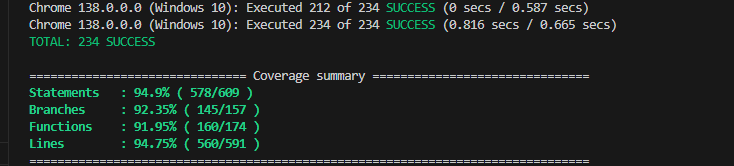
Los cambios realizados a raíz de estas pruebas, llevadas a cabo utilizando Jasmine para los tests unitarios y Karma para la ejecución y medición de cobertura, no fueron extensivos, sino que se enfocaron principalmente en la optimización de la lógica de algunos métodos, mejorando la eficiencia y robustez del código existente.

### **Conclusiones**

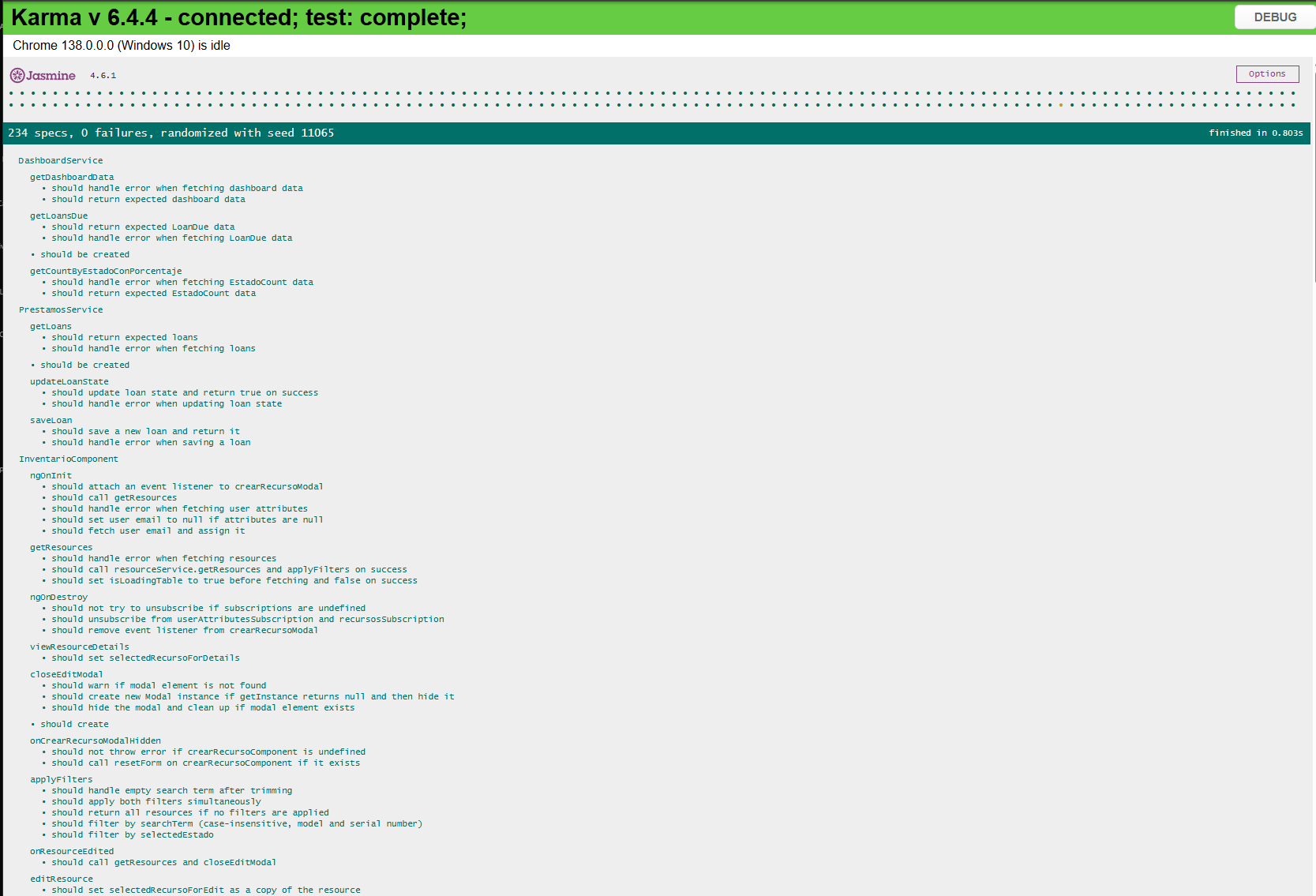
Las pruebas unitarias confirman una solidez considerable y un alto nivel de confianza en la funcionalidad actual del Frontend de Giret. La alta cobertura y la granularidad de los tests aseguran que cada unidad de código se comporta como se espera, minimizando la probabilidad de introducir regresiones.

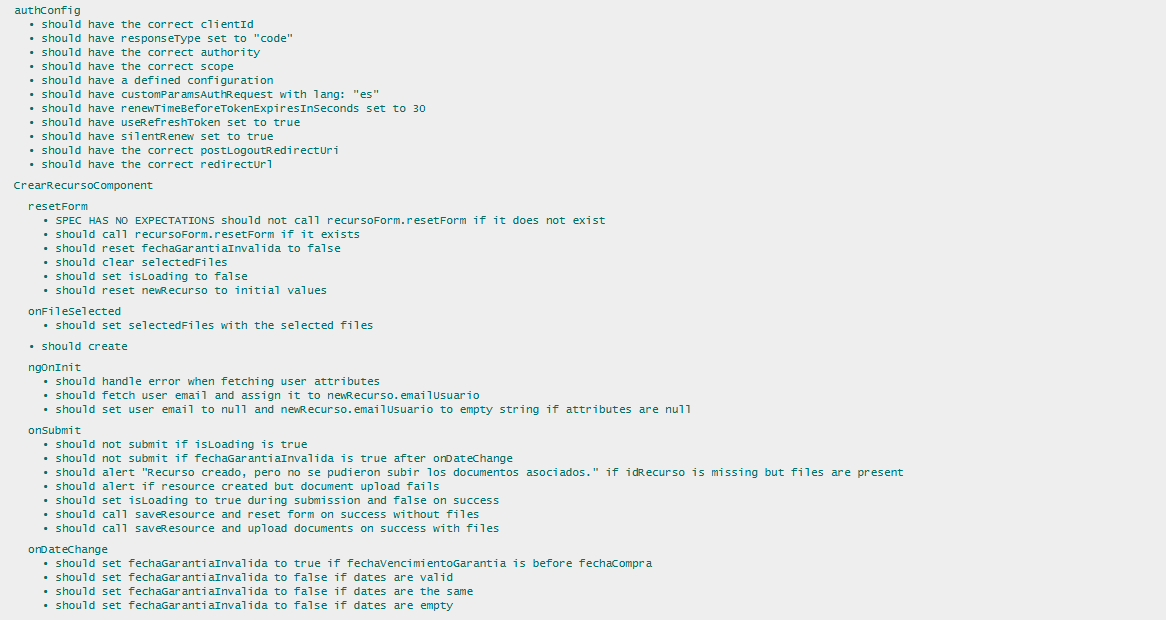
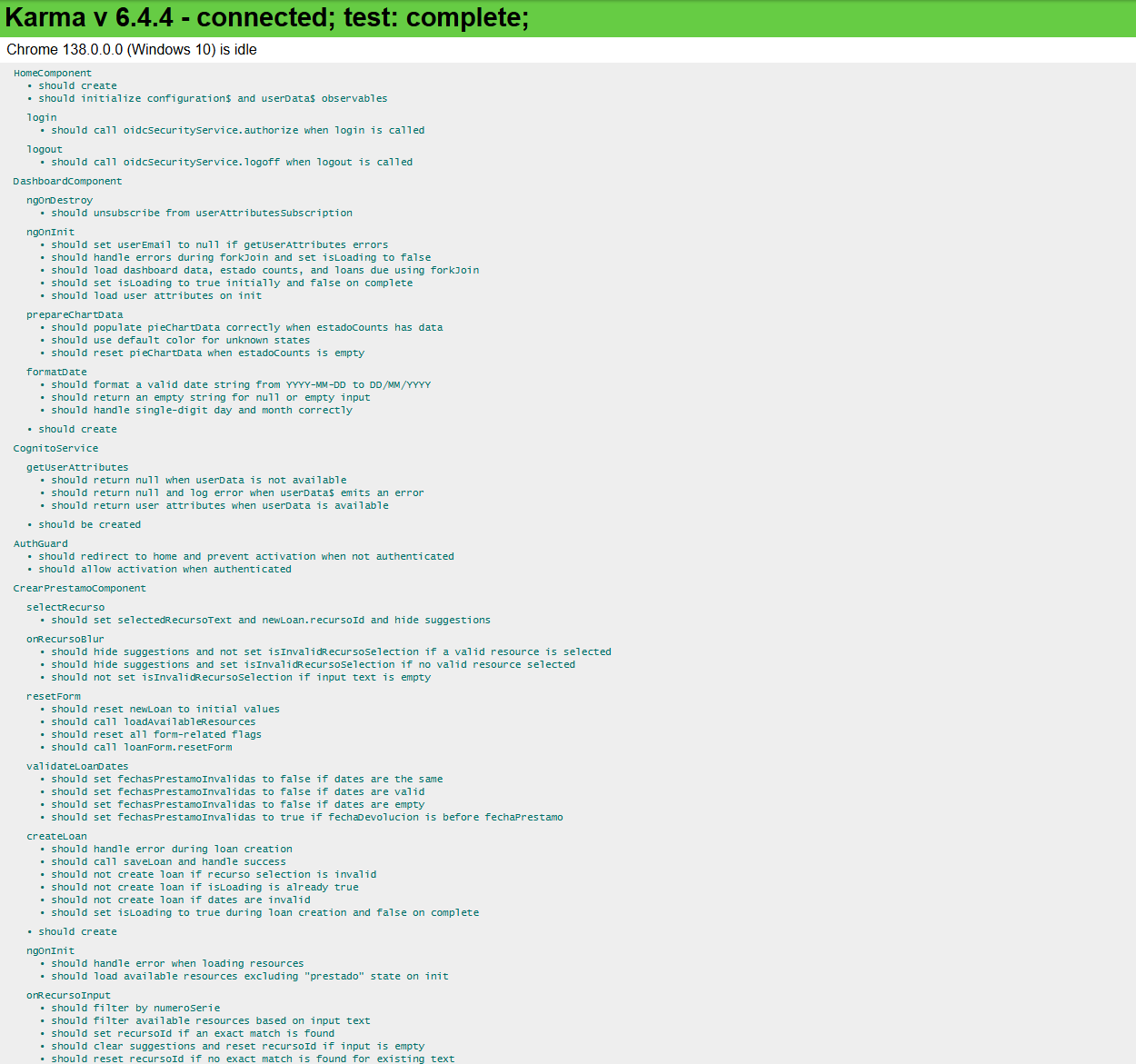
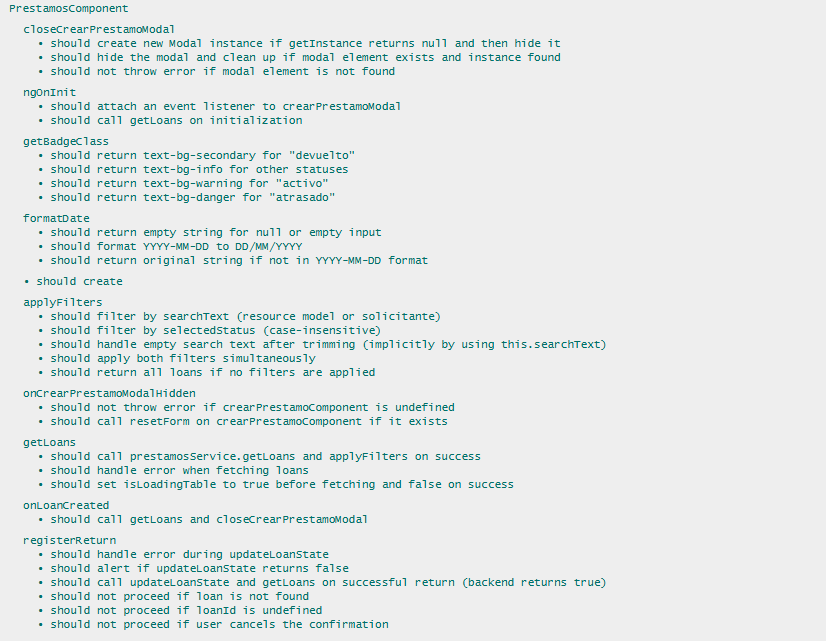
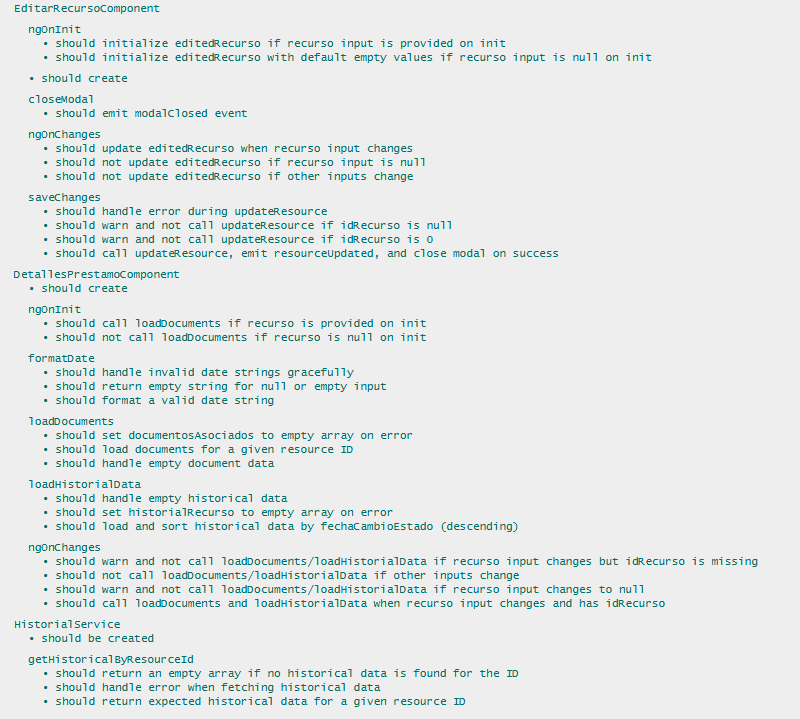
### **Evidencia**

Cobertura de código:



Test Realizados:



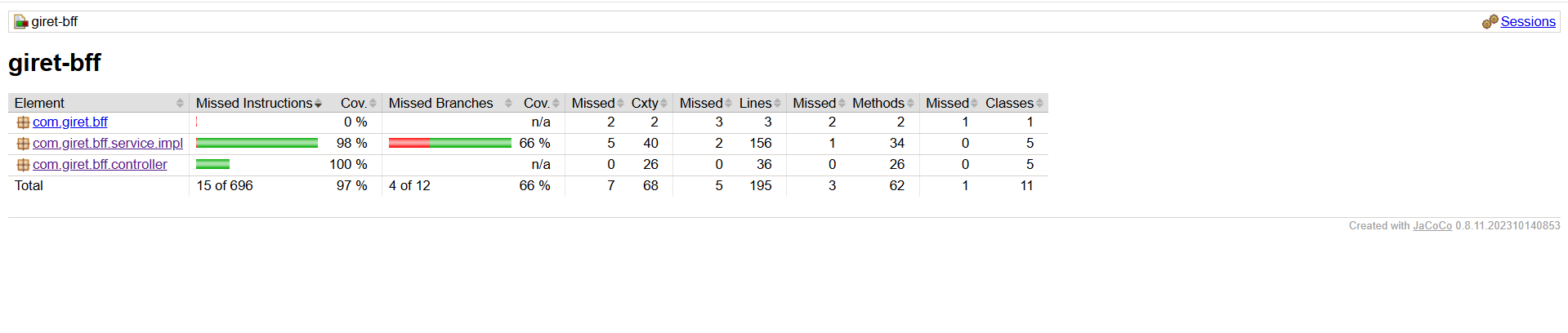


## **10. Cobertura del Código Backend y Calidad del Código**

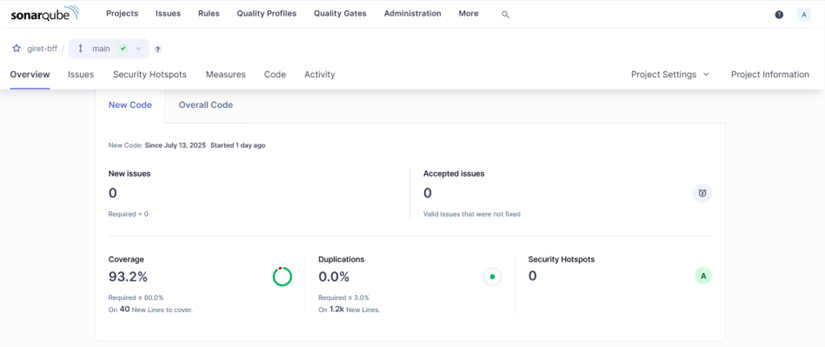
## **Módulo: giret-bff**

Cobertura de Código (JaCoCo):  
 - Instrucciones: 97%  
 - Ramas: 66%  
 - Clases: 11 analizadas (todas alcanzadas).  
 - Detalle por paquete:  
 - com.giret.bff.service.impl: 98% instrucciones, 66% ramas  
 - com.giret.bff.controller: 100% instrucciones  
  
 Análisis de Calidad (SonarQube):  
 - Cobertura global: 93.2%  
 - Code Smells: 0 nuevos detectados.  
 - Hotspots de seguridad: 0  
 - Duplicaciones: 0%  
 - Estado: Alineado al Quality Gate configurado (mínimo 80%).

Evidencia:

A continuación se muestra el reporte JaCoCo del módulo giret-bff:  


Y el tablero SonarQube correspondiente:



## **Módulo: api-giret-resources**

Cobertura de Código (JaCoCo):

- Instrucciones: 97%

- Ramas: 100%

- Clases: 3 analizadas (todas alcanzadas).

- Detalle por paquete:

- com.giret.apigiretresources.service: 100% instrucciones, 100% ramas

- com.giret.apigiretresources.controller: 100% instrucciones

Análisis de Calidad (SonarQube):

- Cobertura global: 100%

- Code Smells: 0 nuevos detectados.

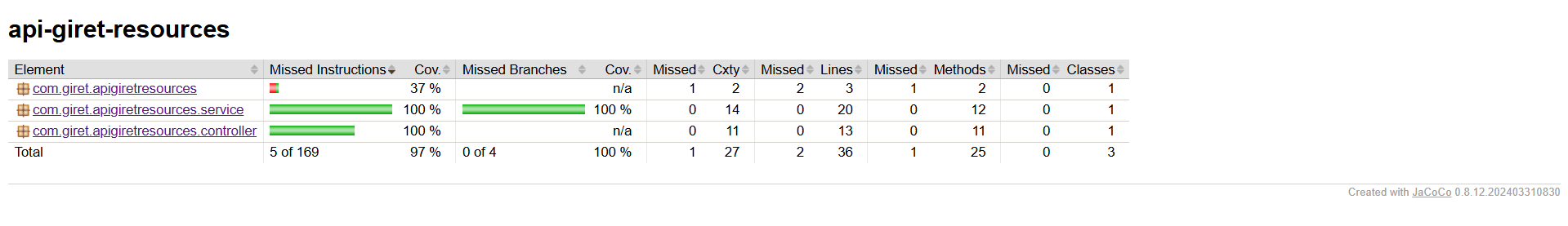
- Hotspots de seguridad: 0

- Duplicaciones: 0%

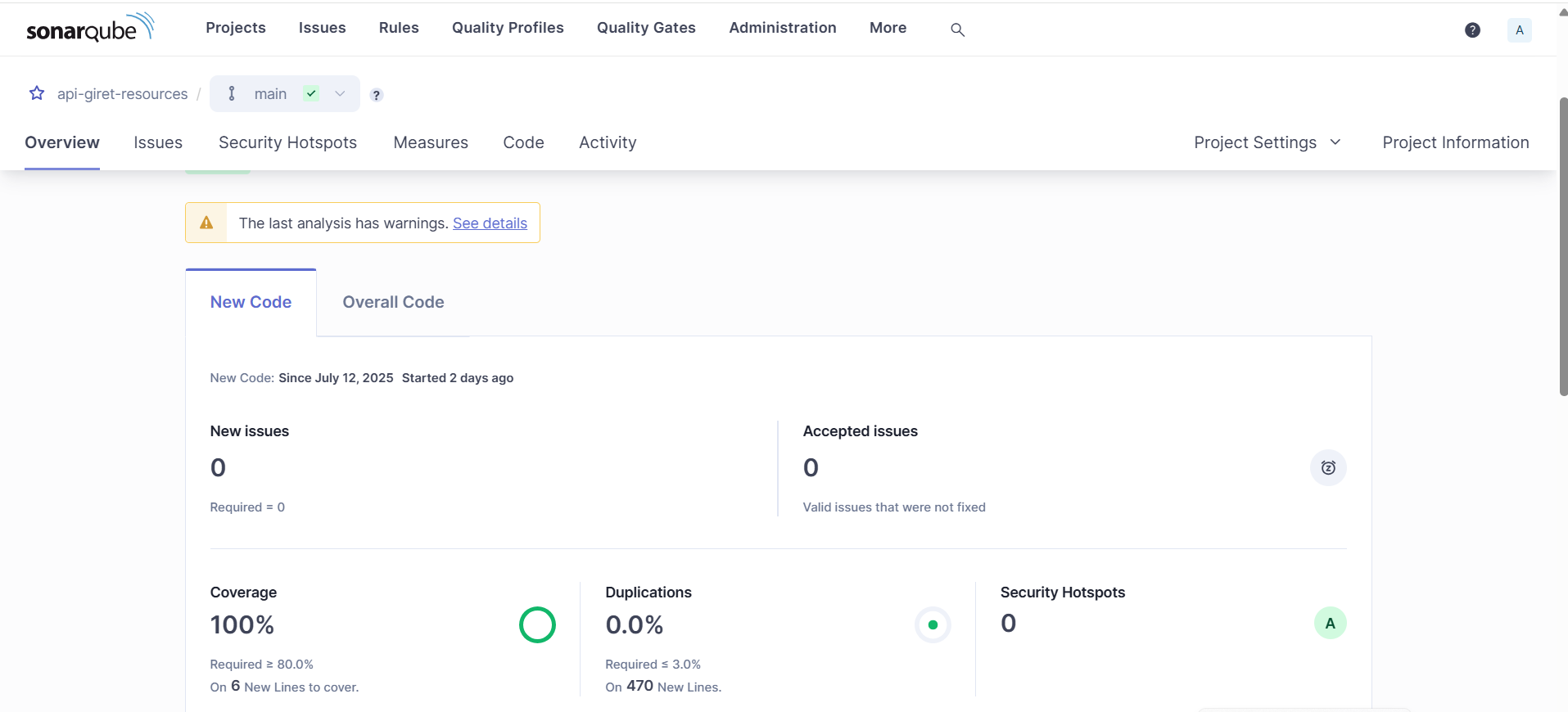
- Estado: Alineado al Quality Gate configurado (mínimo 80%).

Evidencia:

A continuación se muestra el reporte JaCoCo del módulo api-giret-resource



Y el tablero SonarQube correspondiente:



## **Módulo: api-giret-loan**

Cobertura de Código (JaCoCo):

- Instrucciones: 96%

- Ramas: 100%

- Clases: 3 analizadas (todas alcanzadas).

- Detalle por paquete:

- com.giret.loan.service: 100% instrucciones, 100% ramas

- com.giret.loan.controller: 100% instrucciones

Análisis de Calidad (SonarQube):

- Cobertura global: 93.9%

- Code Smells: 0 nuevos detectados.

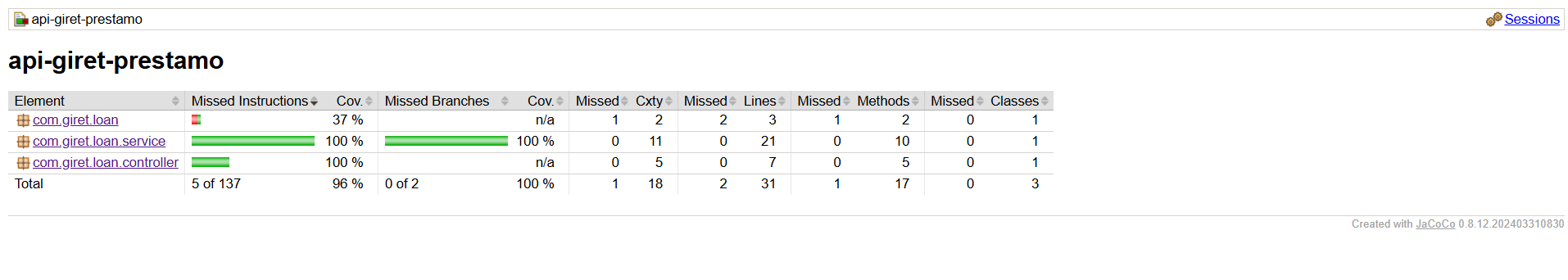
- Hotspots de seguridad: 0

- Duplicaciones: 0%

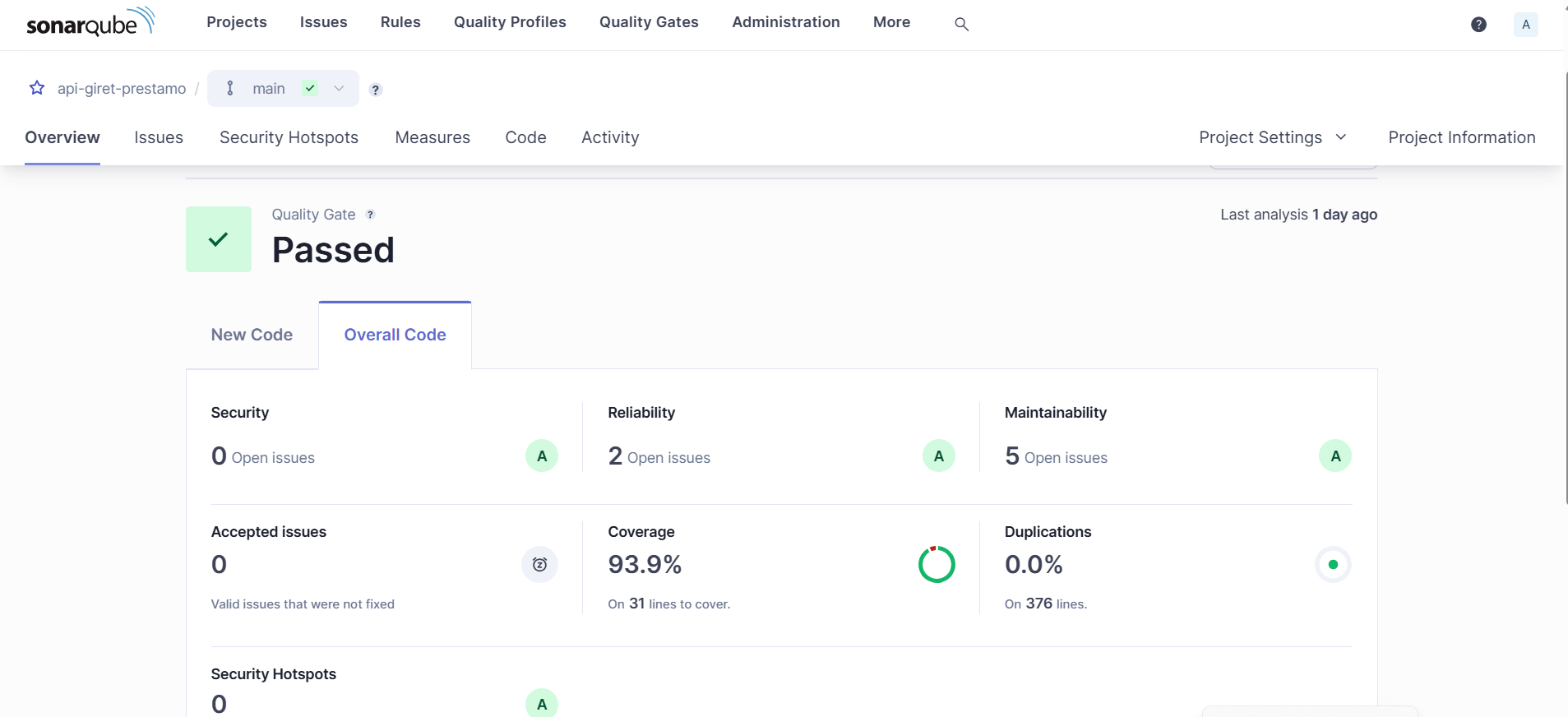
- Estado: Alineado al Quality Gate configurado (mínimo 80%).

Evidencia:

A continuación se muestra el reporte JaCoCo del módulo api-giret-loan



Y el tablero SonarQube correspondiente:



## **Módulo: api-giret-historical-resource**

Cobertura de Código (JaCoCo):

- Instrucciones: 92%

- Ramas: N/A

- Clases: 3 analizadas (todas alcanzadas).

- Detalle por paquete:

- com.giret.apihistorical.resource.service: 100% instrucciones

- com.giret.apihistorical.resource.controller: 100% instrucciones

Análisis de Calidad (SonarQube):

- Cobertura global: 100%

- Code Smells: 0 nuevos detectados.

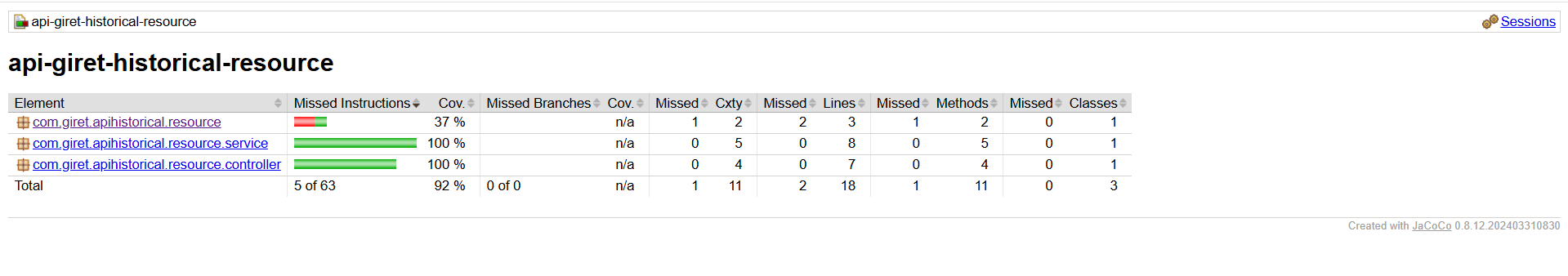
- Hotspots de seguridad: 0

- Duplicaciones: 0%

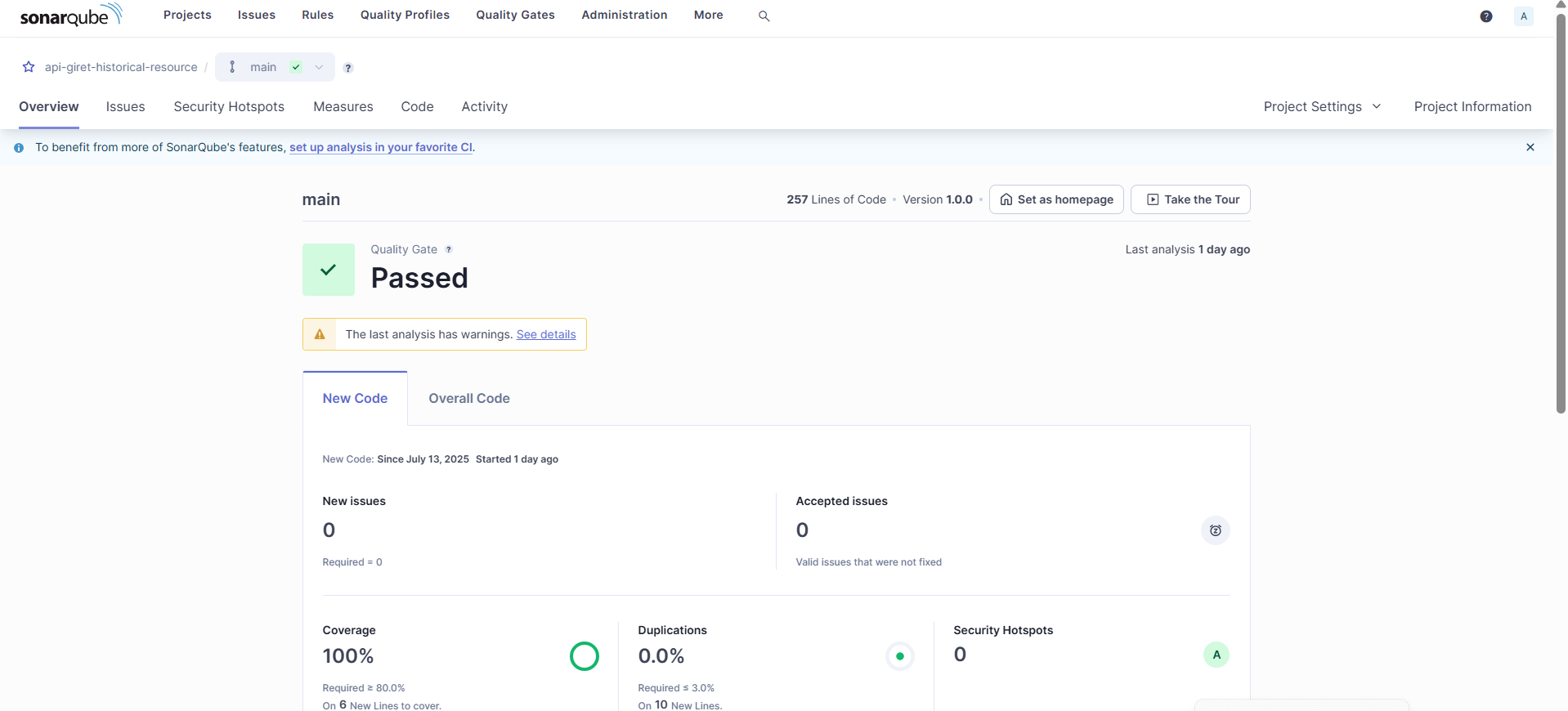
- Estado: Alineado al Quality Gate configurado (mínimo 80%).

Evidencia:

A continuación se muestra el reporte JaCoCo del módulo api-giret-historical-resource



Y el tablero SonarQube correspondiente:



## **Módulo: api-giret-document**

Cobertura de Código (JaCoCo):

- Instrucciones: 88%

- Ramas: 100%

- Clases: 4 analizadas (todas alcanzadas).

- Detalle por paquete:

- com.giret.document.service: 84% instrucciones, 100% ramas

- com.giret.document.controller: 100% instrucciones

- com.giret.document.config: 100% instrucciones

Análisis de Calidad (SonarQube):

- Cobertura global: 100%

- Code Smells: 2 nuevos detectados.

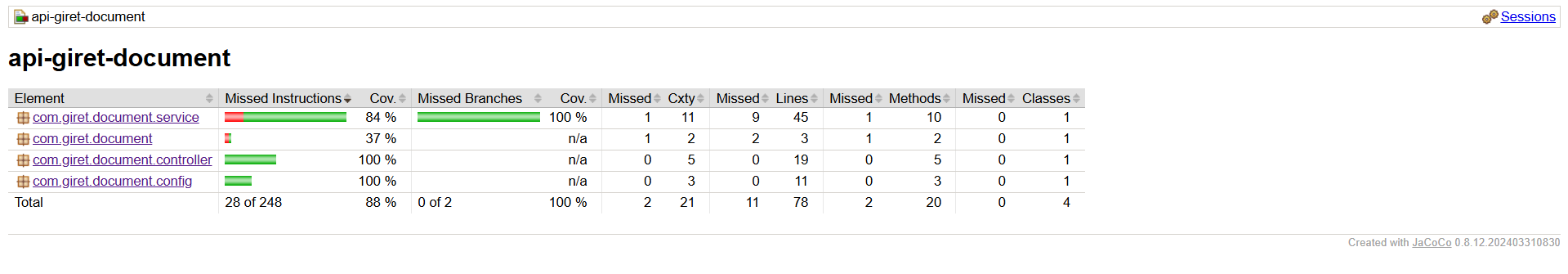
- Hotspots de seguridad: 0

- Duplicaciones: 0%

- Estado: Quality Gate Failed — se requiere corrección de los issues para cumplir con el umbral mínimo del 80%.

Evidencia:

A continuación se muestra el reporte JaCoCo del módulo api-giret-document



Y el tablero SonarQube correspondiente:

