Facultad de
Ciencias Sociales y
Tecnologías de la Información
Talavera de la Reina. UCLM

MEMORIA

CLAUDIA MARTÍN – ANAIS CHIQUITO – DAVID MARCOS – MIGUEL AHIJÓN

INGENIERIA DEL SOFTWARE II

CONTENIDO

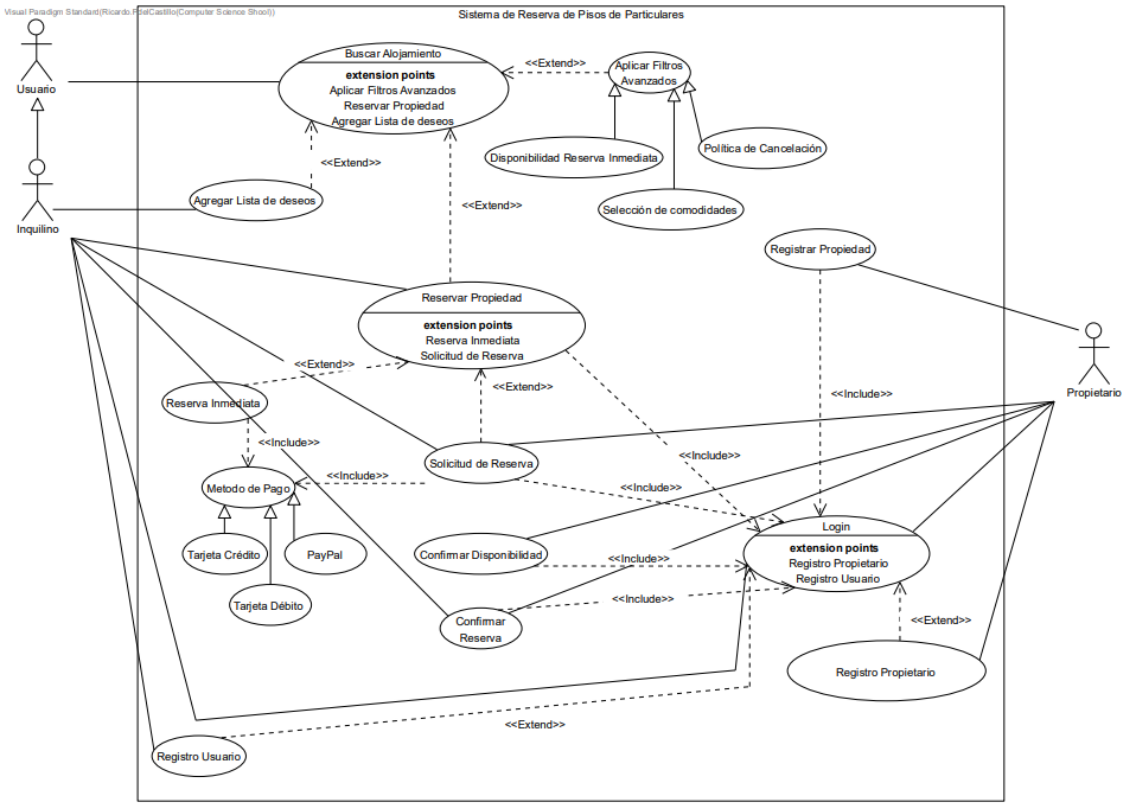
1. INTRODUCCIÓN

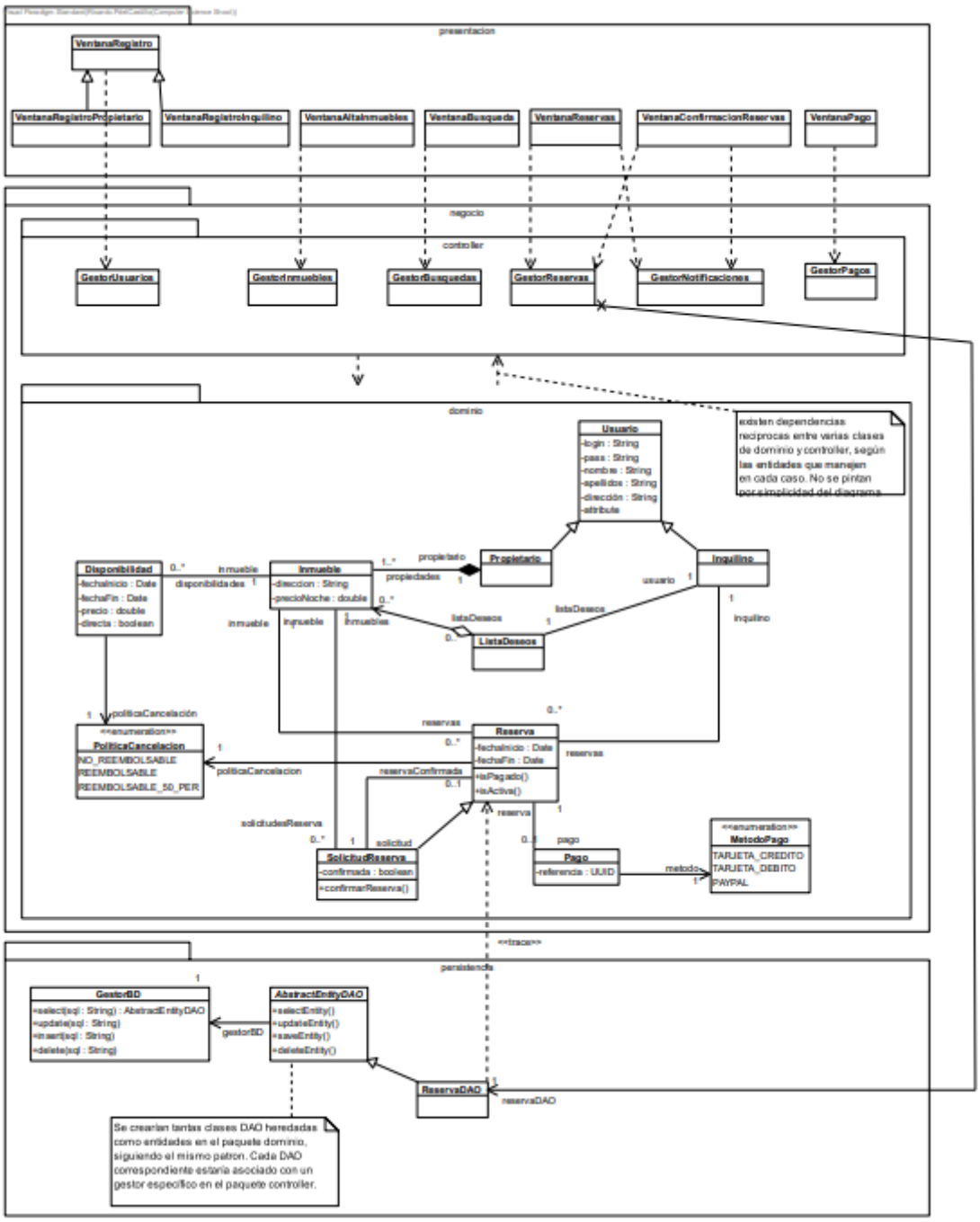
El documento recoge la memoria intermedia del proyecto “HomygoS.L”, desarrollado en la asignatura de Ingeniería del Software II.

El objetivo principal del proyecto es construir una aplicación web para la gestión de alquileres de viviendas entre particulares, similar en conceptos a plataformas como Airbnb, aplicando metodologías agiles y herramientas profesionales de desarrollo colaborativo.

El sistema permite que propietarios registren inmuebles o habitaciones disponibles y que usuarios puedan buscar, reservar y gestionar sus alquileres a través de un entorno web. Además, se contemplan diferentes modos de reserva (inmediata o bajo solicitud), así como el uso de filtros avanzados para mejorar la experiencia de búsqueda.

Esta memoria tiene como propósito presentar los avances logrados durante los primeros sprints, detallando la planificación, metodología empleada, herramientas, estructura técnica del sistema y la distribución de tareas dentro del equipo.





2. PLANIFICACIÓN Y GESTIÓN

2.1. METODOLOGÍA DE DESARROLLO (SCRUM)

El equipo adopto Scrum como metodología principal, organizando el desarrollo en sprints cortos y planificados, con revisiones periódicas.

Las fases de Scrum aplicadas fueron:

1. Sprint Planning: selección de historias de usuario y definición de tareas.
2. Daily Scrum: coordinación diaria del equipo.

3. Sprint Review: evaluación de resultados parciales.
4. Retrospective: análisis de eficiencia y mejoras de proceso.

2.2. ROLES DEL EQUIPO

El equipo cuenta con cuatro integrantes, distribuidos en los siguientes roles:

Rol	Responsabilidad principal
Product Owner	Definir y priorizar el backlog, validar entregas
Srum Master	Asegurar la correcta aplicación de Scrum y eliminar bloqueos
Desarrollador Backend	Implementar la lógica del sistema, entidades y persistencia
Desarrollador Frontend	Desarrollar vistas, controladores y diseño web

2.3. HERRAMIENTAS DE PLANIFICACIÓN

- GitHub Projects: gestión del backlog y tareas mediante tablero Kanban (To Do, In Progress, Done)
- GitHub Milestones: control de hitos y seguimiento por sprints
- Google Docs: documentación
- Maven: soporte de gestión de dependencias y ejecución

2.4. PLANIFICACIÓN POR SPRINTS

El trabajo se estructuro en seis Sprints principales, registrados como milestones en GitHub:

Sprint 1 – Arranque del proyecto y funcionalidades básicas (1 semana)

En este primer sprint se establecieron las bases del proyecto, asegurando que la aplicación arrancase correctamente y que las funcionalidades principales de registro e inicio de sesión funcionasen de forma adecuada. Además, se implementó una persistencia mínima para permitir el almacenamiento básico de datos.

Las tareas principales de este sprint fueron:

- Configuración del entorno de trabajo con Spring Boot
- Creación del proyecto y configuración del pom.xml
- Implementación del registro e inicio de sesión básicos
- Primera conexión con la base de datos (Spring Data JPA)
- Comprobación de funcionamiento de persistencia

Sprint 2 – Desarrollo de vistas y recursos (1 semana)

Durante este sprint se trabajó en el desarrollo de la capa de presentación de la aplicación. Se configuraron los recursos necesarios y se creó la estructura de plantillas HTML para las distintas vistas del sistema, ampliando las funcionalidades disponibles para el usuario.

Las tareas principales de este sprint fueron:

- Configuración de src/main/resources y del archivo application.properties
- Creación de plantillas HTML (login, registro, inmuebles)
- Desarrollo de entidades Usuario, Propietario y Vivienda
- Implementación de controladores y flujo MVC
- Revisión de vistas y estructura básica del sitio

Sprint 3 – Mejora de interfaz y controladores (1 semana)

En este sprint se realizaron mejoras sobre la funcionalidad web existente, reforzando el patrón MVC mediante la ampliación de controladores y la mejora de las vistas HTML. Se añadieron nuevas funcionalidades y se ajustó el comportamiento de la aplicación para una mejor experiencia de usuario.

Las tareas principales de este sprint fueron:

- Optimización de vistas HTML y mejora de usabilidad
- Creación de nuevas clases MVC y refactorización de controladores
- Ampliación de la capa de servicios y conexión entre módulos

Sprint 4 – Persistencia y base de datos (1 semana)

En este sprint se abordó la implementación de la capa de persistencia del proyecto. Se configuró la conexión con la base de datos y se integró Spring Data JPA para la gestión de las entidades del dominio.

Asimismo, se realizaron modificaciones en el archivo pom.xml para incluir las dependencias necesarias y se verificó el correcto funcionamiento de la aplicación con acceso a datos persistentes.

Las tareas principales de este sprint fueron:

- Implementación de repositorios JPA
- Configuración de la base de datos
- Verificación de operaciones CRUD
- Ajustes de configuración en el proyecto Maven

Sprint 5 – Análisis de calidad con SonarCloud (1 semana)

Este sprint estuvo dedicado al análisis de la calidad del código mediante SonarCloud. Se integró la herramienta en el proyecto y se ejecutó un análisis inicial para detectar problemas relacionados con mantenibilidad, fiabilidad y seguridad.

Durante este sprint se realizaron las siguientes tareas:

- Configuración de SonarCloud en el proyecto
- Ejecución del análisis de calidad
- Revisión de code smells, issues y métricas de calidad
- Automatización del análisis mediante GitHub Actions

Este sprint permitió identificar puntos de mejora y planificar acciones de mantenimiento posteriores.

Sprint 6 – Pruebas automáticas y control de calidad (1 semana)

El último sprint se centró en la integración y validación de pruebas automáticas y en el control de calidad del proyecto. Se desarrollaron pruebas unitarias utilizando JUnit y se ejecutaron mediante Maven Surefire. Además, se verificó la correcta ejecución del proyecto mediante procesos de construcción automatizados.

Las tareas realizadas en este sprint incluyeron:

- Implementación de pruebas unitarias con JUnit
- Ejecución automática de tests con Maven Surefire
- Generación de informes de pruebas y cobertura
- Validación final del proyecto mediante builds automáticos

Este sprint permitió consolidar la calidad del software y verificar su correcto funcionamiento antes de la entrega final.

2.5. DISTRIBUCIÓN DE TAREAS POR INTEGRANTE

Integrante	Rol	Funciones
Anaís Chiquito	Product Owner	• Gestión y priorización del backlog • Validación de entregas • Comunicación con el profesor (cliente) • Supervisión general del proyecto
Claudia Martín	Scrum Master	• Coordinación del equipo • Seguimiento del progreso • Eliminación de bloqueos • Control de entregas
David Marcos	Backend Developer	• Programación de la lógica de negocio • Servicios backend • Persistencia y base de datos
Miguel Ahijón	Frontend Developer	• Desarrollo de vistas HTML / Thymeleaf • Integración con controladores • Apoyo en tareas de desarrollo general

3. CODIFICACIÓN

3.1. TECNOLOGÍAS Y ARQUITECTURA

El proyecto se implementó con Spring Boot 3 bajo el lenguaje Java, utilizando el patrón Modelo-Vista-Controlador (MVC), garantizando modularidad y mantenibilidad del código.

Tecnologías principales:

- Spring Boot: framework base para la aplicación
- Spring Data JPA: acceso a base de datos mediante una base de datos H2 embebida, adecuada para entornos de desarrollo y pruebas
- Thymeleaf: motor de plantillas para vistas dinámicas
- Maven: gestor de dependencias y compilación

3.2. COMPONENTES PRINCIPALES

- Modelos: Usuario, Propietario, Vivienda, Reserva
- Servicios: Gestión de reservas, persistencia y validaciones
- Controladores: Coordinación entre las vistas y la capa de negocio
- Vistas: HTML + CSS con Thymeleaf, para login, registro y gestión de propiedades

3.3. AVANCES TÉCNICOS

Durante la entrega intermedia se logró:

- Crear una arquitectura base estable y funcional
- Implementar la comunicación entre capas MVC
- Desarrollar la persistencia con base de datos H2 embebida
- Construir una interfaz funcional de usuario

Se siguieron buenas prácticas de programación:

- Inyección de dependencias (preferiblemente por constructor)
- Comentado y documentación del código
- Manejo de errores y validaciones básicas
- Convenciones de nombres consistentes

4. GESTIÓN DE CONFIGURACIÓN

4.1. CONTROL DE VERSIONES

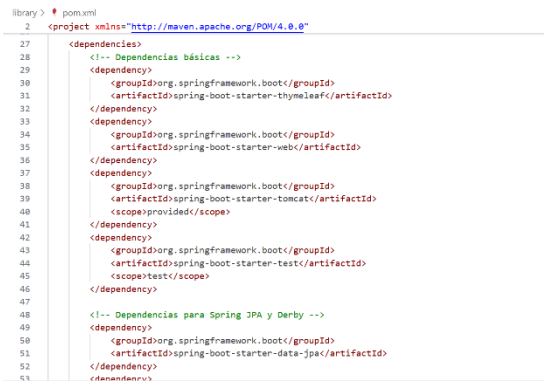
El proyecto se gestiona mediante Git y GitHub, con el flujo de trabajo basado en ramas:

- main: rama estable para entregas
- develop: integración de funcionalidades en curso
- feature/*: ramas especificas por historia de usuario o tarea

Cada commit describe los cambios realizados, y los pull requests garantizan la revisión entre integrantes antes de la integración.

4.2. GESTIÓN DE DEPENDENCIAS Y COMPILACIÓN

El archivo pom.xml contiene las dependencias necesarias del proyecto:



El uso de Maven facilita la construcción del proyecto, ejecución de pruebas unitarias y generación del artefacto .war

4.3. INTEGRACIÓN Y DESPLIEGUE

- Servidor: Tomcat embebido en Spring Boot
- Base de datos: H2 embebida, utilizada para entornos de desarrollo y pruebas, integrada mediante Spring Data JPA
- Pruebas: Validación del login, registro y persistencia de entidades

Se planifica la integración de pruebas automáticas y la futura implementación de un entorno de despliegue remoto para la entrega final.

5. SONAR (ANÁLISIS INICIAL)

Durante esta entrega se ha realizado un análisis estático del código del proyecto HomyGo utilizando la plataforma SonarCloud, con el objetivo de evaluar la calidad, mantenibilidad y fiabilidad del software desarrollado hasta el momento.

5.1. RESULTADOS DEL ANÁLISIS INICIAL

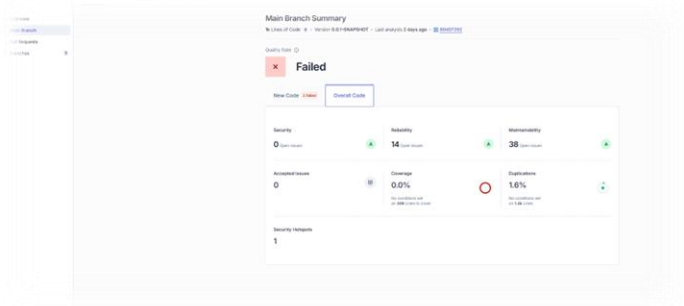
Los resultados obtenidos en la rama main fueron los siguientes:

- Bugs: 0
- Vulnerabilities: 0
- Code Smells: 38
- Security Hotspots: 1
- Maintainability Rating: A
- Reliability Rating: A
- Security Rating: A
- Duplications: 1.6%
- Coverage: 0.0%
- Quality Gate: FAILED

El estado del Quality Gate es FAILED debido a dos condiciones no cumplidas:

- Cobertura insuficiente en código nuevo (0% frente al 80% requerido)
- Ningún security hotspot revisado (requisito al 100%)

Estos resultados muestran que el proyecto no presenta errores críticos, pero si diversas oportunidades de mejora en términos de mantenibilidad.



5.2. PROPUESTA DE CONFIGURACIÓN ALTERNATIVA

En la versión de SonarCloud utilizada no se dispone de permisos de administración para modificar directamente la configuración del análisis. No obstante, se propone una mejora aplicable en un entorno con permisos completos.

La opción que hemos elegido es excluir archivos HTML del análisis

Justificación:

- Los archivos HTML no contienen lógica Java
- Los code smells detectados en estas vistas no aportan información relevante
- Su exclusión permitiría centrar el análisis en el código Java real del proyecto
- Esta práctica es habitual en proyectos con Spring Boot y plantillas Thymeleaf

Pasos teóricos para realizarlo (no aplicables por falta de permisos):

1) Acceder a More → Administration
2) Entrar en General Settings → Analysis Scope
3) Añadir la exclusión anterior en Source File Exclusions
4) Guardar
5) Ejecutar de nuevo el análisis con Maven

5.3. ISSUES SELECCIONADAS Y EXPLICACIÓN

A continuación, se describen tres de los issues más relevantes detectados durante el análisis:

- Issue 1: “Remove this field injection and use constructor injection instead”
- Archivo: Archivo: VentanaAltaInmuebles.java
- Severidad: Major
- Significado: la clase utiliza inyección de dependencias mediante anotación sobre atributos (@Autowired), lo cual dificulta el testeo y oculta dependencias.
- Solución: se recomienda usar inyección por constructor:

```
private final GestorInmuebles gestorInmuebles;

public VentanaAltaInmuebles(GestorInmuebles gestorInmuebles) {

    this.gestorInmuebles = gestorInmuebles;

}
```

- Issue 2: “Replace Stream.collect(Collectors.toList()) with Stream.toList()”
- Archivo: VentanaBusqueda.java
- Severidad: Major
- Significado: uso de una sintaxis antigua de Java para convertir Streams en listas
- Solución: sustituir por la versión moderna -> stream.toList();

- Issue 3: “Replace this instanceof check and cast with 'instanceof Number number'”
- Archivo: ErrorControllerHomyGo.java
- Severidad: Minor
- Significado: uso de patrones antiguos de comprobación y casting

5.4. CONCLUSIONES DEL ANÁLISIS

El análisis realizado con SonarCloud ha permitido identificar aspectos relevantes sobre la calidad del código del proyecto HomyGo.

Aunque no se han detectado bugs ni vulnerabilidades, sí se han registrado diversos code smells que indican posibles mejoras en estilo, mantenibilidad y buenas prácticas.

La propuesta de exclusión de archivos HTML ayudaría a enfocar el análisis únicamente en el código Java, obteniendo resultados más útiles para el equipo de desarrollo.

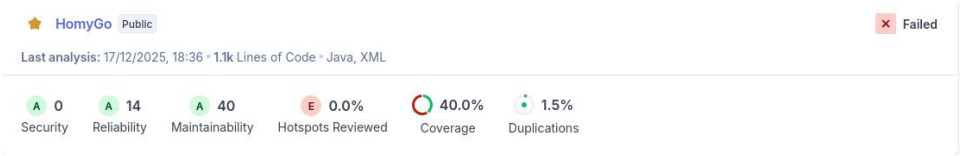
SonarCloud se revela como una herramienta efectiva para garantizar calidad en proyectos colaborativos, detectar errores a tiempo y guiar la mejora continua del código en futuras iteraciones.

5.5. SEGUNDO ANÁLISIS

Tras el análisis inicial, se realizaron mejoras en el proyecto (incorporación de pruebas unitarias y pequeñas refactorizaciones) y se ejecutó un nuevo análisis en SonarCloud para evaluar la evolución de métricas.

En este segundo análisis se observan los siguientes valores:

- Security: 0
- Reliability: 14
- Maintainability: 40
- Coverage: 40.0%
- Duplications: 1.5%
- Quality Gate: FAILED



5.6. PRUEBAS UNITARIAS (JUNIT)

5.6.1. INTRODUCCIÓN A LAS PRUEBAS UNITARIAS

Para mejorar la fiabilidad del proyecto y aumentar la cobertura se añadieron pruebas unitarias con JUnit 5, cubriendo principalmente:

- Servicios/gestores (lógica de negocio)
- Entidades del dominio (getters/setters y comportamiento básico)
- Controladores con pruebas de tipo MVC

La ejecución de pruebas se realiza con Maven:

- mvn test
- mvn clean test

5.6.2. TESTS IMPLEMENTADOS (2 EXPLICADOS)

Para la validación del sistema se han definido casos de prueba basados en clases de equivalencia y comprobación de condiciones límite, siguiendo un enfoque sistemático de testing. A continuación, se describen los principales casos de prueba implementados.

5.6.2.1. TEST DEL GESTORUSUARIOS

Testing:

A) Método: registrarUsuario(Usuario usuario)

Caso	Login	Pass	findByLogin(login)	Resultado esperado	Verificación adicional
CP1	"ana"	"1234"	null	Registra y devuelve el usuario	Se llama a findByLogin y save
CP2	null	"1234"	-	Excepción: "El login es obligatorio"	No hay interacciones con repositorio
CP3	""	"1234"	-	Excepción: "El login es obligatorio"	No hay interacciones con repositorio
CP4	"ana"	null	-	Excepción: "La contraseña es obligatoria"	No hay interacciones con repositorio
CP5	"ana"	""	-	Excepción: "La contraseña es obligatoria"	No hay interacciones con repositorio
CP6	"ana"	"1234"	Usuario existente	Excepción: "Ya existe un usuario con ese login"	Se llama a findByLogin y save

B) Método: validarLogin(String login, String pass)

Caso	Usuario en repo	Contraseña guardada	Contraseña introducida	Resultado esperado
CP7	No existe (null)	-	"1234"	false
CP8	Existe	"abcd"	"1234"	false
CP9	Existe	"1234"	"1234"	true

C) Método: listarUsuarios()

Caso	findAll() devuelve	Resultado esperado
CP10	Lista con 2 usuarios	Lista tamaño 2

En este test se valida la lógica de negocio de la clase GestorUsuarios, encargada de la gestión de usuarios del sistema. El objetivo principal es comprobar que las principales operaciones relacionadas con los usuarios se comportan correctamente y que se aplican las validaciones definidas en la lógica del proyecto.

Para ello, se simula el comportamiento del repositorio UsuarioRepository mediante Mockito, evitando así la dependencia de una base de datos real y permitiendo que el test sea completamente unitario.

El test verifica los siguientes casos:

- Registro correcto de un usuario cuando el login y la contraseña son válidos y no existe previamente otro usuario con el mismo login
- Lanzamiento de una excepción cuando el login o la contraseña son nulos o están vacíos
- Lanzamiento de una excepción cuando se intenta registrar un usuario con un login ya existente
- Validación del método validarLogin, comprobando los casos de usuario inexistente, contraseña incorrecta y contraseña correcta
- Comprobación del método listarUsuarios, verificando que devuelve correctamente la lista proporcionada por el repositorio

Además, se comprueba que los métodos del repositorio (findByLogin, save y findAll) solo se invocan cuando corresponde, asegurando que no se persisten datos inválidos y que la lógica de negocio se ejecuta correctamente.

Este test permite garantizar la fiabilidad de la lógica de negocio relacionada con la gestión de usuarios y contribuye a la estabilidad general del sistema.

5.6.2.2. TEST DE LA ENTIDAD LISTADESEOS

Testing:

Caso	Acción / Entrada	Resultado esperado
CP1	Crear new ListaDeseos()	usuario == null, inmuebles no nulo y vacío, id == 0
CP2	setUsuario(inquilino)	getUsuario() devuelve exactamente ese inquilino
CP3	setInmuebles(lista con 1 inmueble)	getInmuebles() devuelve la lista y su tamaño es 1
CP4	setInmuebles(lista vacía)	getInmuebles() no nulo y vacío
CP5	setUsuario(null)	getUsuario() queda en null sin error

Este test se centra en verificar el correcto funcionamiento de la entidad ListaDeseos, perteneciente al modelo de dominio del proyecto. Al tratarse de una entidad JPA, el objetivo principal del test es asegurar la correcta inicialización de sus atributos y el funcionamiento adecuado de sus métodos de acceso, garantizando un estado consistente del objeto desde su creación.

En concreto, el test comprueba:

- Que el constructor por defecto inicializa correctamente la lista de inmuebles y deja el usuario asociado a null, evitando posibles errores de tipo NullPointerException
- El correcto funcionamiento de los métodos get y set para el usuario asociado a la lista de deseos, incluyendo la asignación de valores nulos sin provocar errores
- La correcta asignación y recuperación de la lista de inmuebles, tanto cuando contiene elementos como cuando se trata de una lista vacía

Este tipo de pruebas resulta especialmente útil para detectar errores de inicialización y asegurar la consistencia de las entidades del dominio antes de su integración con la capa de persistencia, contribuyendo a la estabilidad general del sistema.

5.7. INFORMES DE PRUEBAS Y COBERTURA (SUREFIRE / JACOCO)

Además de ejecutar las pruebas, se generaron informes automáticos:

- Surefire: informe de ejecución de tests

- JaCoCo: informe de cobertura

Comandos utilizados:

- mvn clean test
- mvn site (para generar los informes HTML en target/site)

Estos informes permiten verificar de forma objetiva la mejora de cobertura reflejada en SonarCloud.

5.8. MANTENIMIENTO DEL SOFTWARE

A partir de los resultados de SonarCloud se realizaron tareas de mantenimiento orientadas a:

- Reducir deuda técnica y mejorar mantenibilidad
- Aplicar recomendaciones de calidad (por ejemplo, favorecer inyección por constructor frente a inyección por campo)
- Asegurar que el proyecto es verificable automáticamente mediante tests y análisis

El plan de mantenimiento definido para el proyecto se ha llevado a cabo de forma efectiva durante las últimas iteraciones del desarrollo. Dicho plan se centró en la mejora de la calidad del código, la reducción de deuda técnica y la validación automática del sistema.

Las tareas de mantenimiento planificadas y ejecutadas fueron las siguientes:

- Ejecución de un análisis inicial de calidad mediante SonarCloud
- Identificación de problemas de mantenibilidad y ausencia de pruebas unitarias
- Implementación de pruebas unitarias con JUnit para servicios y entidades
- Configuración de la ejecución automática de pruebas mediante Maven Surefire
- Refactorización puntual del código siguiendo recomendaciones de SonarCloud
- Ejecución de un segundo análisis de SonarCloud para evaluar la evolución del sistema

Tras la ejecución del plan de mantenimiento, el estado del sistema muestra una mejora clara en los indicadores de calidad. En particular, se observa un incremento significativo de la cobertura de código, pasando de un 0% en el análisis inicial a aproximadamente un 40% tras la incorporación de pruebas unitarias.

En términos de calidad, el proyecto mantiene un número nulo de bugs y vulnerabilidades, y conserva calificaciones altas en fiabilidad y mantenibilidad. La comparación entre ambos análisis permite concluir que el mantenimiento realizado ha tenido un impacto positivo en la calidad global del sistema, especialmente en lo relativo a la validación automática y control de calidad del código.

5.9. COMPARATIVA DE MÉTRICAS

Métrica	Análisis inicial	Segundo análisis
Code Smells	38	40
Coverage	0.0%	40.0%
Duplications	1.6%	1.5%
Quality Gate	FAILED	FAILED

La principal mejora observable es el aumento de cobertura tras incorporar pruebas unitarias, y una ligera reducción en duplicación.

6. CONCLUSIÓN

Durante la entrega intermedia, el proyecto Homygo S.L. ha cumplido con los objetivos establecidos en las tres áreas evaluables del contrato:

- Planificación y gestión: trabajo iterativo siguiendo Scrum, con planificación real en GitHub y roles definidos.
- Codificación: estructura sólida bajo arquitectura MVC con componentes funcionales.
- Gestión de configuración: uso eficiente de Git, ramas, Maven y control de dependencias.
- Además, se ha incorporado un proceso de calidad (SonarCloud) y pruebas (JUnit + informes) que permite evaluar de forma objetiva la evolución del proyecto.
- El segundo análisis refleja la mejora tras las acciones de mantenimiento realizadas.

El equipo mantiene una colaboración activa y ha logrado construir una base sólida para la entrega final, donde se completarán las funcionalidades de reservas, pagos y optimización del sistema.