**Caso de estudio: Aplicación de gestión de préstamos de libros (SRS)**

*Asignatura: Ingeniería del Software II · Versión 1.0 · 13/10/2025*

# 1. Introducción

Este documento especifica los requisitos software (funcionales y no funcionales) de una aplicación básica de gestión de préstamos de libros para una biblioteca. Servirá como base para el diseño, la implementación y la verificación del sistema.

# 2. Alcance

El sistema permitirá gestionar el fondo bibliográfico, los usuarios y el ciclo de vida de los préstamos (altas, renovaciones, devoluciones y multas por retrasos). Quedan fuera de alcance: adquisición de ejemplares, pasarelas de pago externas, y analítica avanzada.

# 3. Actores y roles

**• Bibliotecario/a:** Gestiona libros, usuarios y préstamos.

**• Usuario/a lector/a:** Consulta el catálogo y gestiona sus préstamos (vía mostrador o portal).

# 4. Supuestos y restricciones

• Identificación de usuarios mediante DNI/NIE o ID institucional.  
• Cada ejemplar tiene un identificador único (código de barras/RFID opcional).  
• Multas calculadas por día de retraso según política configurable.  
• Sistema desplegado como aplicación web con base de datos relacional.

# 5. Requisitos funcionales (RF)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código | Descripción | Prioridad | Tiempo estimado | Dependencias |
| RF.01 | Dar de alta libros en el fondo bibliográfico con metadatos mínimos (título, autor, ISBN, edición, año, materia, número de ejemplares). | Alta | 3 días | - |
| RF.02 | Actualizar información de libros (metadatos, estado, ejemplares disponibles) con control de versiones/cambios. | Alta | 2 días | RF.01 |
| RF.03 | Consultas de libros por múltiples criterios (título, autor, ISBN, materia, disponibilidad) con paginación y ordenación. | Alta | 3 días | RF.01 |
| RF.04 | Dar de baja libros (lógico/soft delete) con validación de que no existan préstamos activos vinculados. | Alta | 1.5 días | RF.01, RF.11 |
| RF.05 | Dar de alta usuarios con datos de contacto y categoría (estudiante, PDI/PAS, externo) y políticas de préstamo asociadas. | Alta | 2 días | - |
| RF.06 | Actualizar información de usuarios (datos personales, estado activo/inactivo, categoría). | Alta | 1.5 días | RF.05 |
| RF.07 | Dar de baja usuarios (lógico) con verificación de préstamos/multas pendientes. | Alta | 1.5 días | RF.05, RF.11, RF.12 |
| RF.08 | Consultas de usuarios por criterios (nombre, DNI/NIE, email, categoría, estado). | Media | 1.5 días | RF.05 |
| RF.09 | Dar de alta un préstamo: asignar ejemplar disponible a usuario, registrar fechas (inicio, vencimiento) y política aplicada. | Alta | 3 días | RF.01, RF.05 |
| RF.10 | Actualizar un préstamo (renovar, registrar devolución, cambiar fecha por causas justificadas) con reglas de negocio. | Alta | 3 días | RF.09 |
| RF.11 | Dar de baja (cerrar) un préstamo al registrarse la devolución; liberar ejemplar y actualizar disponibilidad. | Alta | 2 días | RF.09, RF.10 |
| RF.12 | Asociar una multa por retraso al préstamo: cálculo automático por días de retraso y registro de la deuda del usuario. | Alta | 2 días | RF.09, RF.11 |
| RF.13 | Notificaciones: avisar por email de vencimiento próximo y de retrasos. | Media | 2 días | RF.05, RF.09 |
| RF.14 | Histórico y trazabilidad de cambios en libros, usuarios y préstamos (auditoría básica). | Media | 2 días | RF.02, RF.06, RF.10 |

# 6. Requisitos no funcionales (RNF)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código | Descripción | Prioridad | Tiempo estimado | Dependencias |
| RNF.01 | Autenticación y control de acceso por roles (bibliotecario, lector), con sesiones seguras. | Alta | 3 días | RF.05 |
| RNF.02 | Protección de datos conforme RGPD: consentimiento, minimización de datos, derechos ARCO, y cifrado en tránsito (TLS 1.2+). | Alta | 3 días | RF.05 |
| RNF.03 | Disponibilidad 99,5% mensual en horario laboral (L-V 8:00–20:00). | Media | — | - |
| RNF.04 | Rendimiento: consultas comunes < 1s p95 con base de datos de hasta 50k libros y 20k usuarios. | Media | 2 días (optimización) | RF.03, RF.08 |
| RNF.05 | Escalabilidad horizontal básica (stateless app) y conexión a BD escalable. | Media | 2 días | - |
| RNF.06 | Usabilidad: interfaz web responsive, accesible (WCAG 2.1 AA), formularios con validación y mensajes claros. | Media | 3 días | RF.01–RF.12 |
| RNF.07 | Auditabilidad: registro de eventos críticos (altas/bajas/cambios) con usuario, fecha y detalle. | Media | 1.5 días | RF.14 |
| RNF.08 | Backup diario y restauración probada (RPO ≤ 24h, RTO ≤ 4h). | Media | 1 día | - |
| RNF.09 | API interna documentada (OpenAPI) para integrar un futuro portal del lector. | Baja | 2 días | RF.03, RF.09 |
| RNF.10 | Mantenibilidad: cobertura de pruebas ≥ 70% en lógica de negocio y linters/formatters en CI. | Media | 3 días | - |
| RNF.11 | Internacionalización (i18n) preparada; textos externalizados. | Baja | 1 día | - |
| RNF.12 | Registro y gestión de configuración centralizada por entorno (dev/test/prod). | Baja | 1 día | - |

# 7. Trazabilidad (alto nivel)

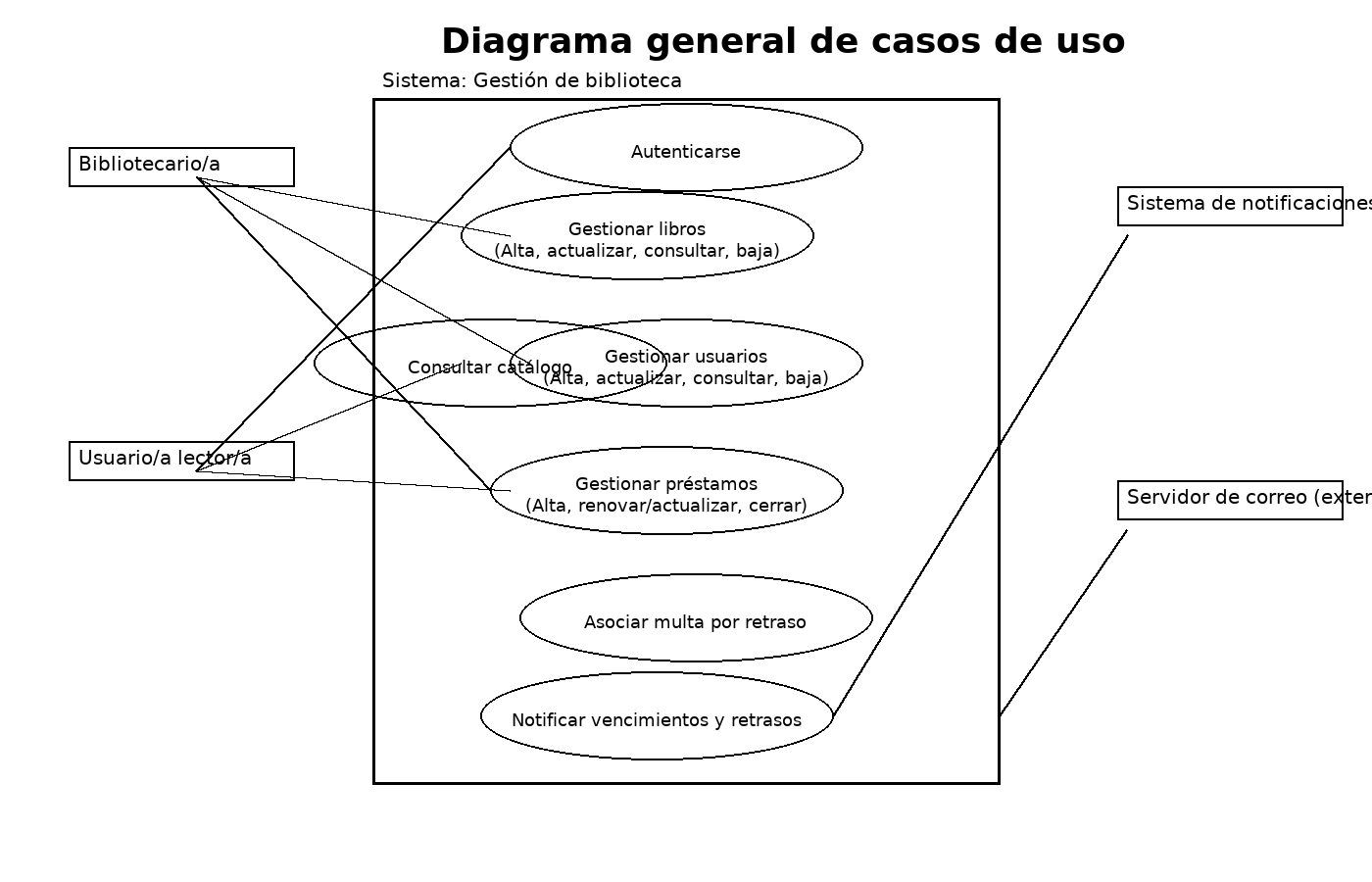
• Gestión de fondos bibliográficos → RF.01–RF.04  
• Gestión de usuarios → RF.05–RF.08  
• Gestión de préstamos y multas → RF.09–RF.12  
• Soportes transversales → RF.13–RF.14, RNF.\*

# 8. Criterios de aceptación generales

1) Todos los flujos RF.01–RF.12 disponen de pruebas funcionales que cubren casos nominales y de error.  
2) Cumplimiento RNF.02 (RGPD) verificado mediante checklist.  
3) Rendimiento RNF.04 medido con dataset de prueba representativo.  
4) Auditoría RNF.07 registra usuario, timestamp y entidad afectada.

## 5.x Vista de requisitos: Diagrama general de casos de uso

El siguiente diagrama muestra los actores principales y su asociación con los casos de uso a alto nivel.

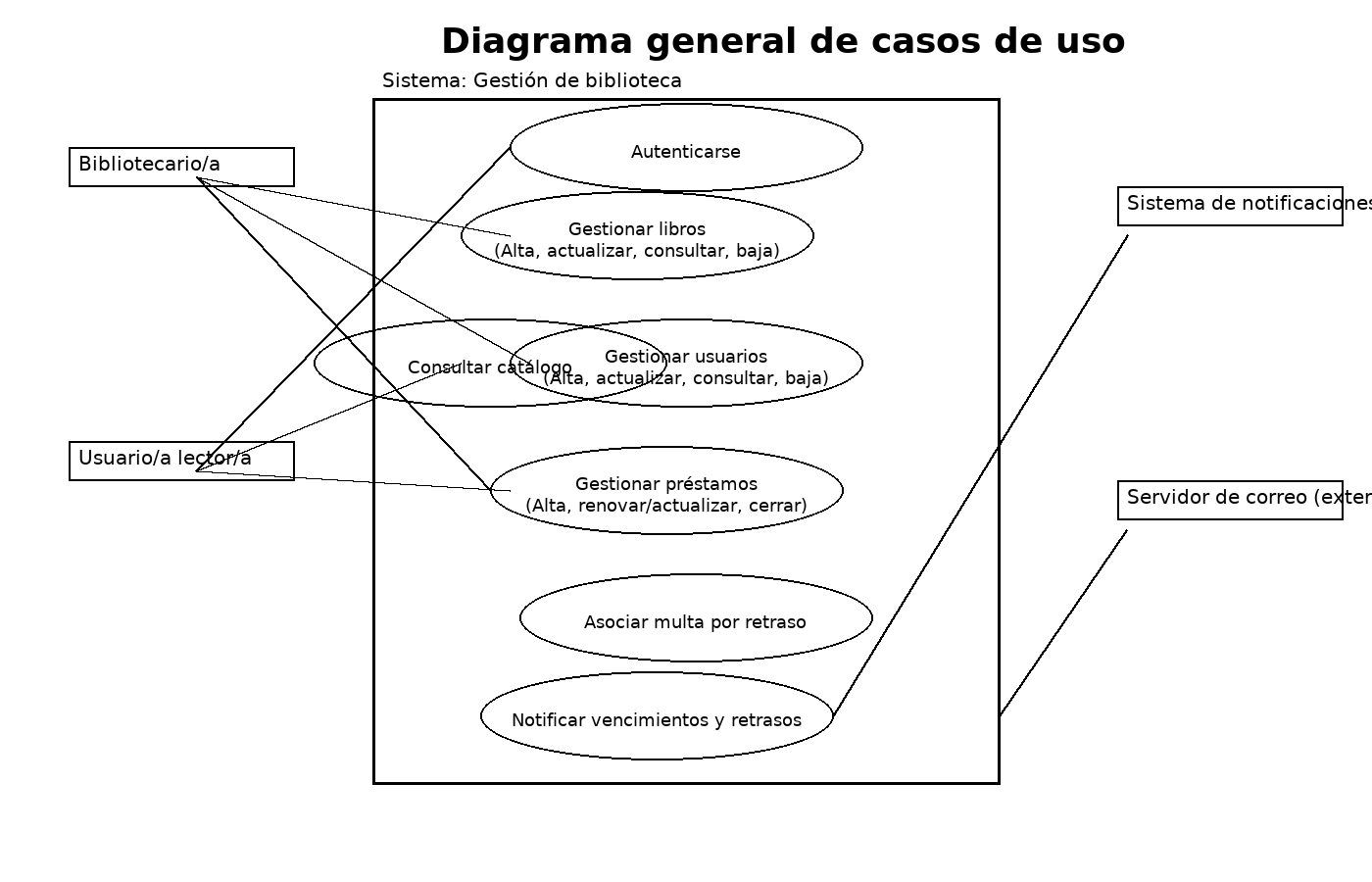


# 9. Análisis del sistema (MVC monolítico multicapa)

## 9.1 Actores y casos de uso

Actores principales:  
• Bibliotecario/a: ejecuta casos de uso de gestión (libros, usuarios, préstamos y multas).  
• Usuario/a lector/a: consulta catálogo, solicita/renueva préstamos y recibe notificaciones.  
• Sistema de notificaciones y servidor de correo: sistemas externos integrados para envío de avisos.

Diagrama general de casos de uso (ver imagen):



## 9.2 Clases de análisis y estereotipos

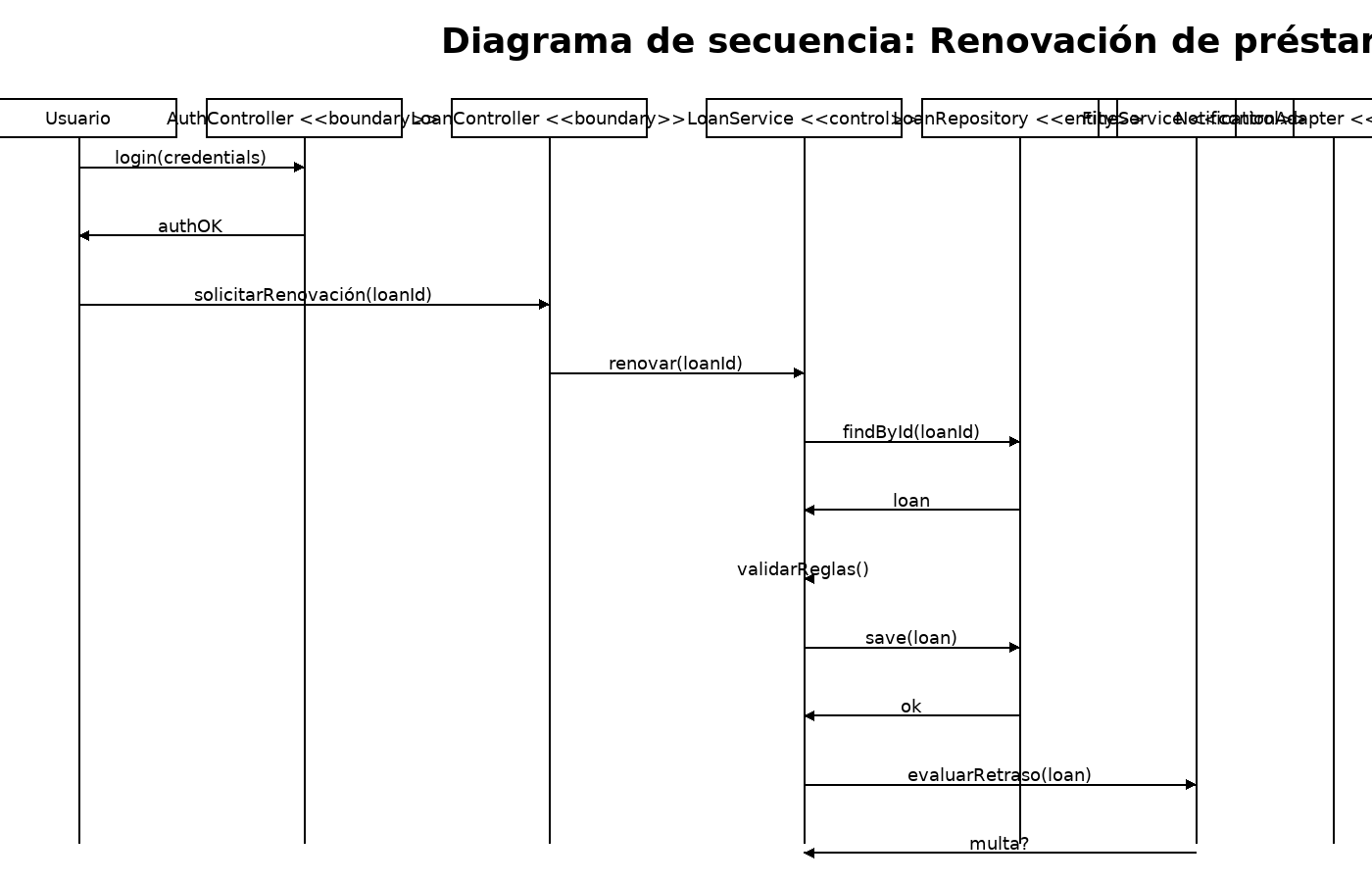
Se identifican las clases de análisis con estereotipos UML según su rol: <<boundary>>, <<control>>, <<entity>>.

|  |  |  |
| --- | --- | --- |
| Clase | Estereotipo | Responsabilidad principal |
| AuthController | <<boundary>> | Gestionar autenticación y sesiones. |
| BookController | <<boundary>> | Endpoints/vistas para gestión de libros. |
| UserController | <<boundary>> | Endpoints/vistas para gestión de usuarios. |
| LoanController | <<boundary>> | Endpoints/vistas para préstamos y renovaciones. |
| NotificationAdapter | <<boundary>> | Interfaz con el servidor de correo. |
| CatalogService | <<control>> | Orquestación de consultas del catálogo. |
| UserService | <<control>> | Reglas de negocio de usuarios. |
| LoanService | <<control>> | Reglas de negocio de préstamos (renovaciones, cierres). |
| FineService | <<control>> | Cálculo de multas por retrasos. |
| Book | <<entity>> | Entidad de libro con metadatos. |
| Copy | <<entity>> | Ejemplar físico asociado a un libro. |
| User | <<entity>> | Datos del usuario y estado de sanciones. |
| Loan | <<entity>> | Préstamo con fechas, estado y renovaciones. |
| Fine | <<entity>> | Multa vinculada a un préstamo y a un usuario. |

Notas:  
• Los control coordinan transacciones y aplican reglas; las boundary exponen el sistema a actores o externos; las entity encapsulan estado persistente.  
• Repositorios/DAOs se consideran parte de la infraestructura y pueden modelarse como <<boundary>> hacia la capa de datos o omitirse a nivel de análisis.

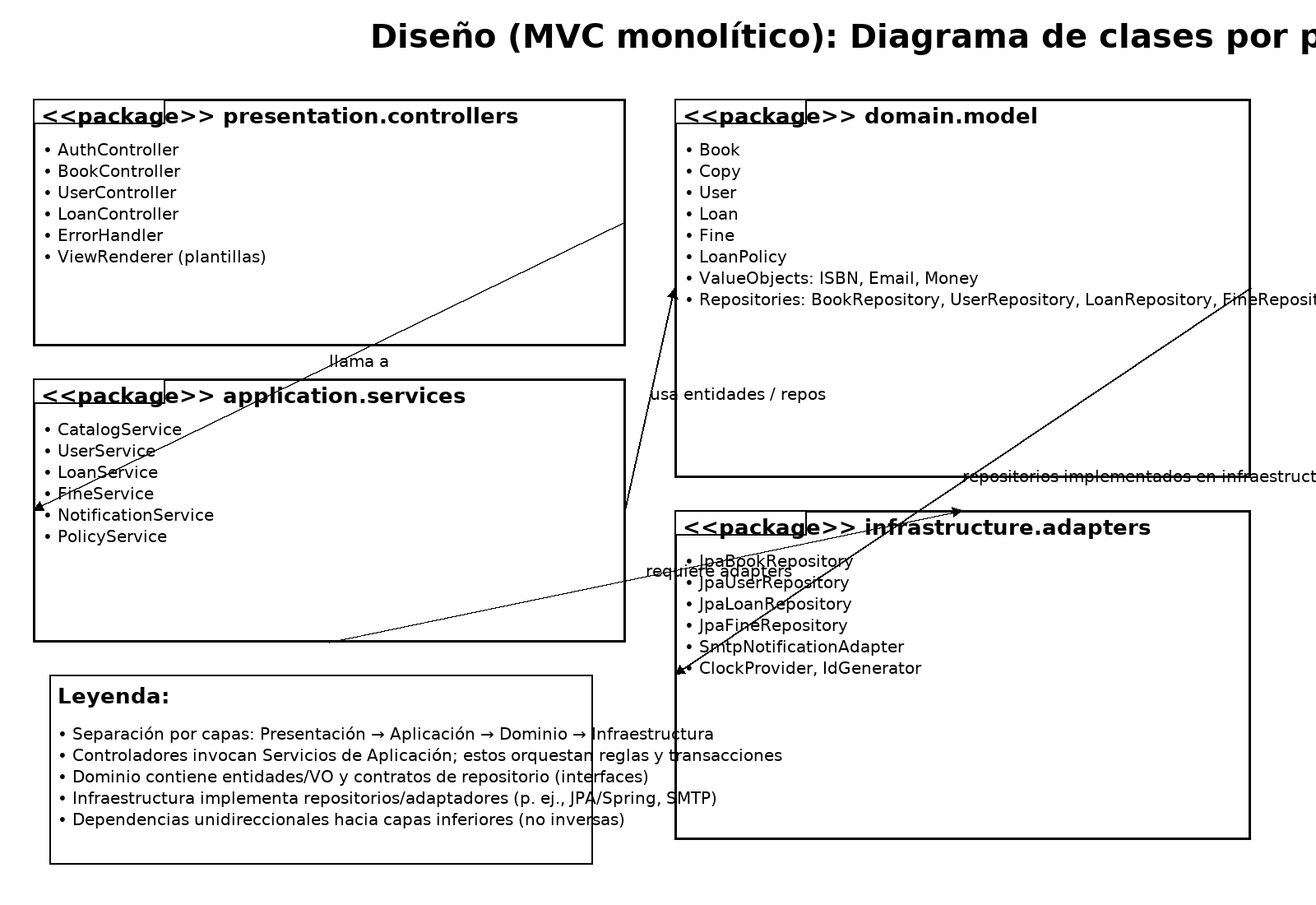
## 9.3 Diagrama de secuencia (ejemplo: renovación de préstamo)

Este diagrama ilustra la interacción típica para una renovación de préstamo iniciada por un usuario autenticado. Incluye validaciones de reglas de negocio, persistencia y notificaciones.



# 10. Diseño: Diagrama de clases por paquetes (MVC)

Se presenta un diagrama de clases de diseño organizado por paquetes y capas, siguiendo el patrón MVC en un monolito multicapa. Incluye controladores (presentación), servicios de aplicación, modelo de dominio (entidades y objetos de valor), y adaptadores/infraestructura.



## 10.1 Mapa de clases por paquete

|  |  |  |
| --- | --- | --- |
| Paquete | Clases | Notas de diseño |
| presentation.controllers | AuthController, BookController, UserController, LoanController, ErrorHandler, ViewRenderer | Reciben peticiones, validan entrada superficial y delegan en servicios. |
| application.services | CatalogService, UserService, LoanService, FineService, NotificationService, PolicyService | Aplican casos de uso, gestionan transacciones e invocan repositorios/adaptadores. |
| domain.model | Book, Copy, User, Loan, Fine, LoanPolicy, VO: ISBN, Email, Money; Repos: BookRepository, UserRepository, LoanRepository, FineRepository | Entidades ricas con invariantes; repositorios definidos como interfaces. |
| infrastructure.adapters | Jpa\*Repository, SmtpNotificationAdapter, ClockProvider, IdGenerator | Implementaciones técnicas (JPA/SMTP/etc.). No contienen reglas de dominio. |

## 10.2 Notas de separación de capas

• Los controladores jamás acceden directamente a la persistencia.  
• Los servicios orquestan reglas (p. ej., cálculo de multas, políticas por categoría) y coordinan repositorios.  
• Las entidades encapsulan lógica y validan invariantes (e.g., no renovar si hay reserva o multa activa).  
• Las implementaciones de repositorio/adaptador se inyectan vía interfaces para aislar el dominio.

# 11. Implementación (estructura de proyecto y convenciones)

Se ha generado un proyecto Maven (archetype 'maven-archetype-quickstart') con groupId='es.uclm.esi.iso2' y artifactId='bibliotecamonolitica', empaquetado 'jar'. La estructura sigue el patrón MVC con separación en paquetes conforme al GroupID.

## 11.1 Estructura de paquetes (conforme al GroupID)

es.uclm.esi.iso2.bibliotecamonolitica.  
├── presentation.controllers  
├── application.services  
├── domain.model  
├── domain.repositories  
├── infrastructure.adapters  
├── common.exceptions  
└── common.util

## 11.2 Convenciones de código y validaciones en setters

Se han implementado getters/setters en todas las entidades; los setters de atributos String validan cadenas vacías lanzando NotNullValueAllowedException. Los constructores usan los setters para asegurar validaciones homogéneas.

## 11.3 Ejecución y manifiesto

El empaquetado 'jar' incluye manifiesto con Main-Class = es.uclm.esi.iso2.bibliotecamonolitica.Main. Ejemplo de ejecución:  
 mvn -q -DskipTests package  
 java -jar target/bibliotecamonolitica-1.0.0.jar

## 11.4 Documentación Javadoc

Se configura maven-javadoc-plugin para generar documentación:  
 mvn javadoc:javadoc  
El resultado se genera en target/site/apidocs.

## 11.5 Alineación pom.xml ↔ paquetes

El POM define el groupId/artifactId y la ruta de paquetes base (es.uclm.esi.iso2.bibliotecamonolitica). Las clases se ubican bajo src/main/java/es/uclm/esi/iso2/bibliotecamonolitica/... asegurando coherencia con el GroupID.

## 11.6 Dependencias y configuración para MySQL, pruebas y javadoc

Se han añadido al pom.xml las dependencias y plugins necesarios:  
• MySQL: mysql-connector-j (driver JDBC) para comunicación con la base de datos.  
• Pruebas: junit-jupiter-api y junit-jupiter-engine (JUnit 5) + maven-surefire-plugin para su ejecución.  
• Documentación: maven-javadoc-plugin para generar la API en HTML.

Configuración de entorno para MySQL (leída por AgentMySQL):  
 MYSQL\_URL=jdbc:mysql://localhost:3306/biblioteca?useSSL=false&serverTimezone=UTC  
 MYSQL\_USER=usuario  
 MYSQL\_PASSWORD=secreto  
Scripts SQL en src/main/resources/sql: schema.sql (crea tablas) y seed.sql (100 usuarios y 100 libros).

## 11.7 Agente de BD y DAOs (patrón Singleton + DAO)

Se introduce un agente abstracto (Agent) con patrón Singleton y su implementación concreta AgentMySQL. Las clases DAO (BookDAO, UserDAO, LoanDAO, FineDAO) usan Agent.getInstance().getConnection() para realizar operaciones JDBC. Los servicios de aplicación ahora delegan en DAOs para persistencia y consultas.

# 12. Pruebas

Se han diseñado e implementado pruebas unitarias con JUnit 5 para cubrir el 90% o más del código. Se incluye el plugin JaCoCo para medir y \*\*verificar\*\* la cobertura mínima (regla ≥ 90% a nivel de bundle).

## 12.1 Alcance y criterios

• Utilidades y validaciones (Validation, NotNullValueAllowedException).  
• Objetos de valor (ISBN, Email, Money).  
• Entidades (Book, User, Loan, Fine) — pruebas de setters/getters e invariantes.  
• Servicios (CatalogService, UserService) — pruebas con DAOs sustituidos por implementaciones in situ.  
• DAOs (BookDAO, UserDAO) — pruebas de integración contra H2 en memoria en modo MySQL, cargando schema.sql.

## 12.2 Ejecución e informes

Ejecución: `mvn -q test` genera el informe de cobertura en `target/site/jacoco/index.html`. La fase `verify` falla si la cobertura de líneas es inferior al 90%.

## 12.3 Preparación de BBDD en pruebas

Se emplea un agente de pruebas `AgentH2` (solo en test) que configura H2 en modo MySQL. Los tests cargan `/sql/schema.sql` mediante `DBTestSupport.runSqlResource(...)`. Esto evita dependencias externas y acelera la ejecución manteniendo compatibilidad SQL.

# 13. Instrucciones para la creación del proyecto en Visual Studio Code y conexión con GitHub

## 13.1 Instalación y configuración inicial

1. Instalar Visual Studio Code (VS Code): Descargar desde https://code.visualstudio.com e instalarlo. Durante la instalación, marcar la opción 'Agregar a PATH'.  
2. Instalar extensiones necesarias:  
 - Extension Pack for Java (Microsoft)  
 - Maven for Java  
 - Debugger for Java  
 - GitHub Pull Requests and Issues (opcional)  
 - GitLens (para historial de commits y comparación de versiones)  
3. Instalar Git: Descargar desde https://git-scm.com/downloads. Verificar la instalación con el comando 'git --version' en la terminal.

## 13.2 Creación del proyecto Maven

1. Abrir VS Code y ejecutar el comando desde la paleta (Ctrl+Shift+P): 'Maven: Create Maven Project'.  
2. Seleccionar el archetype 'maven-archetype-quickstart'.  
3. Introducir los valores:  
 - Group ID: es.uclm.esi.iso2  
 - Artifact ID: bibliotecamonolitica  
 - Version: 1.0.0  
4. Abrir el proyecto generado en VS Code.  
5. Sustituir el contenido del código fuente con los archivos del proyecto proporcionado.

## 13.3 Compilación y ejecución

En la terminal integrada de VS Code, ejecutar:  
 mvn clean package  
 java -jar target/bibliotecamonolitica-1.0.0.jar

## 13.4 Conexión con Git y GitHub

1. Inicializar un repositorio Git:  
 git init  
 git add .  
 git commit -m 'Primer commit: versión inicial del proyecto bibliotecamonolitica'  
2. Crear un nuevo repositorio en GitHub (https://github.com/new) con el nombre 'bibliotecamonolitica'. No marcar 'Initialize with a README'.  
3. Conectar el repositorio local con el remoto:  
 git remote add origin https://github.com/<tu\_usuario>/bibliotecamonolitica.git  
 git branch -M main  
 git push -u origin main  
4. Configurar las credenciales de GitHub utilizando un token personal de acceso (https://github.com/settings/tokens).

## 13.5 Uso de Git dentro de Visual Studio Code

1. Abrir la vista de 'Control de Código Fuente' en la barra lateral izquierda.  
2. Escribir un mensaje de commit y presionar el icono ✓ o Ctrl+Enter para confirmar.  
3. Pulsar 'Sync Changes' o ejecutar 'git push' para subir los cambios a GitHub.

## 13.6 Flujo de trabajo recomendado

1. Crear una rama para cada nueva funcionalidad:  
 git checkout -b feature/nueva-funcionalidad  
2. Realizar los cambios y commits:  
 git add .  
 git commit -m 'Implementación de nueva funcionalidad'  
3. Subir los cambios y crear un Pull Request en GitHub para fusionarlos en main.

# 14. Matriz de trazabilidad y Completitud Funcional (ISO/IEC 25010)

La completitud funcional se calcula como el cociente entre requisitos funcionales implementados y requisitos especificados. Se detalla la trazabilidad entre requisitos (RF), clases/modulos y pruebas unitarias asociadas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Código | Descripción resumida | Implementado | Clases/Módulos | Pruebas | Estado |
| RF.01 | Alta de libros | Sí | BookController; CatalogService; BookDAO | BookDAOTest; CatalogServiceTest | Verificado |
| RF.02 | Actualizar información de libros | Sí | BookController; BookDAO | BookDAOTest | Verificado |
| RF.03 | Consultas de libros | Sí | BookController; CatalogService | BookControllerTest | Verificado |
| RF.04 | Baja de libros | Sí | BookDAO; CatalogService | BookDAOTest | Verificado |
| RF.05 | Alta de usuarios | Sí | UserController; UserService; UserDAO | UserDAOTest; UserServiceTest | Verificado |
| RF.06 | Actualizar información de usuarios | Sí | UserDAO; UserService | UserServiceTest | Verificado |
| RF.07 | Baja de usuarios | Sí | UserDAO | UserDAOTest | Verificado |
| RF.08 | Consultas de usuarios | Sí | UserController; UserService | UserServiceTest | Verificado |
| RF.09 | Alta de préstamo | Sí | LoanService; LoanDAO | LoanDAOTest; LoanServiceTest | Verificado |
| RF.10 | Renovación/actualización de préstamo | Sí | LoanService | LoanServiceTest | Verificado |
| RF.11 | Cierre/baja de préstamo | Sí | LoanService; LoanDAO | LoanServiceTest | Verificado |
| RF.12 | Asociar multa por retraso | Sí | FineService; FineDAO | FineDAOTest | Verificado |

Completitud Funcional = (12 implementados / 12 especificados) × 100 = 100 %.