Patrones de arquitectura de software

JORGE QUINTERO

NELSON CONTRERAS

Taller 1

Principios SOLID

ING. Jairo Sehoanes

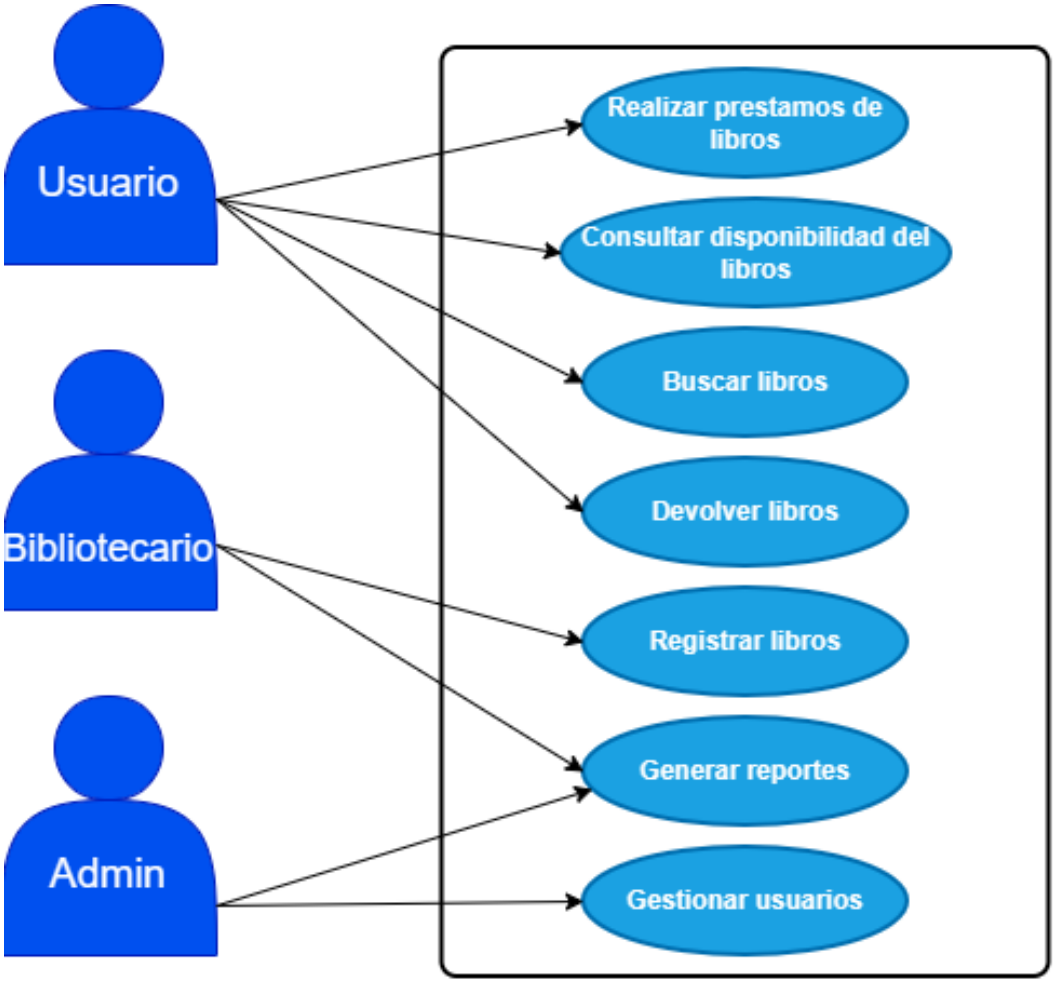
UNIVERSIDAD POPULAR DEL CESAR

FACULTAD DE INGENIERIAS Y TECNOLÓGIAS

ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE

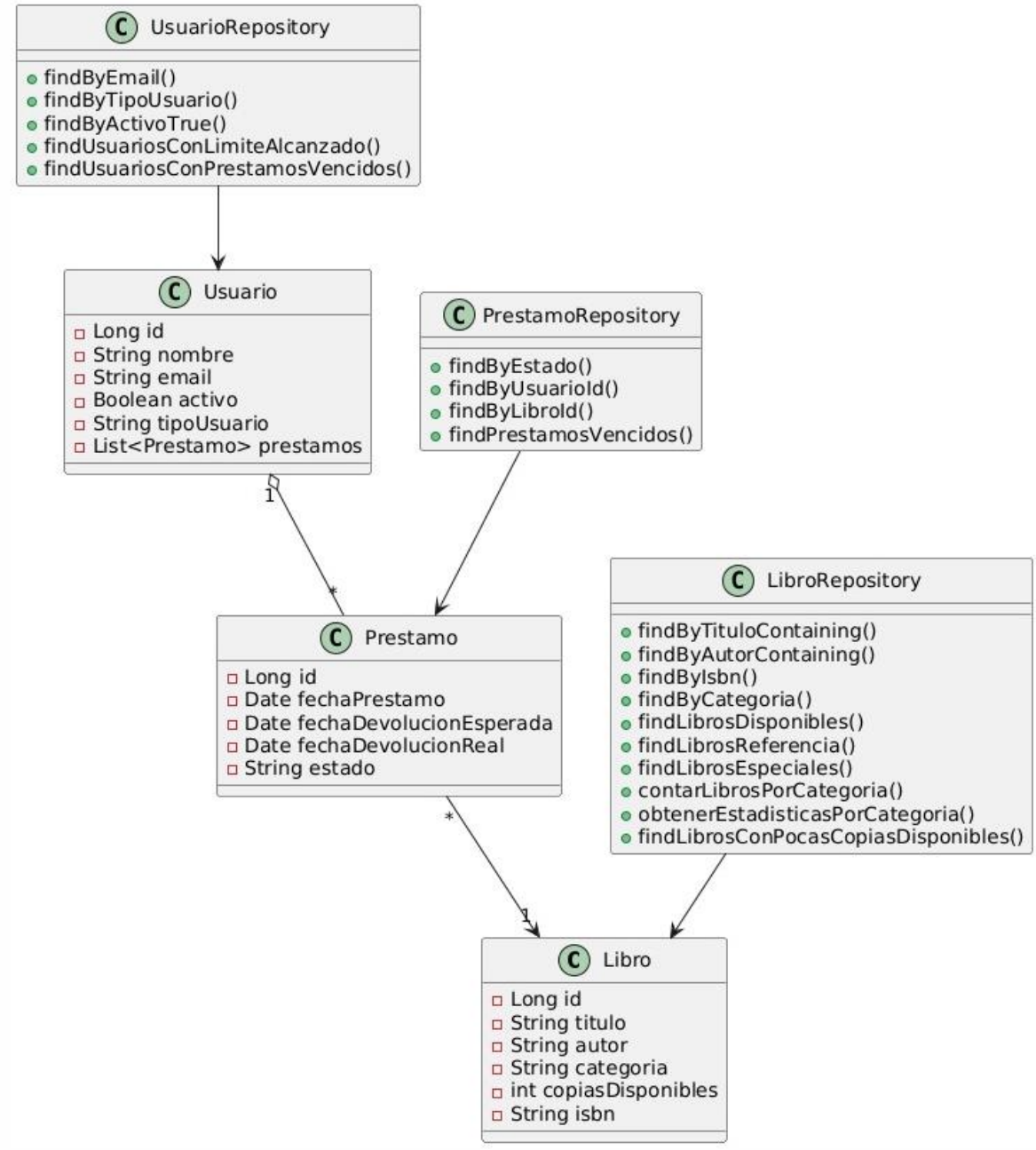
Análisis funcional

Diagrama de casos de uso inicial



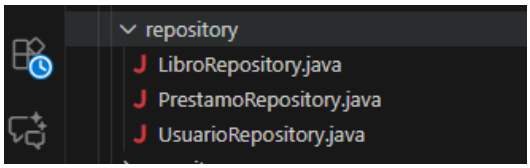
Análisis estructural

Diagrama de clases del estado actual del modelo de dominio



Hallazgos problemas de diseño capa de repositorio

Clases: Biblioteca repository



1. Diagnóstico de Violaciones SOLID

Tras un análisis exhaustivo del código fuente **LibroRepository**, se han detectado violaciones críticas en tres de los cinco principios SOLID. A continuación, se detalla cada hallazgo.

A. Violación del Principio de Responsabilidad Única (SRP)

- **Definición:** Una clase o interfaz debe tener una sola razón para cambiar.
- **Hallazgo:** La interfaz LibroRepository actual tiene múltiples razones para cambiar, asumiendo responsabilidades que no le corresponden:
    1. **Persistencia:** Gestión de datos (CRUD).
    2. **Reportes:** Métodos analíticos (contarLibrosPorCategoria, obtenerEstadisticas).
    3. **Lógica de Negocio:** Define reglas arbitrarias dentro de las consultas SQL (ej. < 2 copias).
- **Impacto:** Si cambia la definición de "pocas copias" o se requiere un nuevo formato de reporte, se debe modificar el repositorio, arriesgando la estabilidad de las funciones CRUD básicas.

B. Violación del Principio de Segregación de Interfaces (ISP)

- **Definición:** Los clientes no deben depender de interfaces que no utilizan.
- **Hallazgo:** La interfaz está sobrecargada ("fat interface"). Un servicio que solo necesite buscar por ISBN se ve obligado a depender de una interfaz que contiene métodos de reporte complejos y búsquedas específicas de "Libros de Referencia".
- **Impacto:** Aumenta el acoplamiento y la complejidad innecesaria para los clientes de la API.

C. Violación del Principio Abierto/Cerrado (OCP)

- **Definición:** Las entidades deben estar abiertas a la extensión pero cerradas a la modificación.
- **Hallazgo:** El uso de consultas con valores fijos ("hardcoded") como WHERE l.categoria = 'REFERENCIA' obliga a modificar el código fuente cada vez que se requiera buscar una nueva categoría (ej. "REVISTAS").
- **Impacto:** Código rígido y poco mantenible.

2. Plan de Cambios (Línea por Línea)

Se instruye realizar las siguientes modificaciones en el archivo original:

Bloque de Código (Líneas aprox.)	Acción	Justificación Técnica
findLibrosDisponibles (Líneas 10-11)	**Modificar**	Cambiar a findByCopiasDisponiblesGreaterThan(int val). Eliminar el valor 0 fijo.
findLibrosReferencia (Líneas 13-14)	**Eliminar**	Redundante. Se debe usar findByCategoria(String c) pasando "REFERENCIA" como parámetro.
findLibrosEspeciales (Líneas 16-17)	**Eliminar**	Redundante. Viola OCP al fijar la categoría en la consulta.
contarLibrosPorCategoria (Líneas 20-21)	**Mover**	Extraer a una nueva interfaz LibroEstadisticaRepository (Segregación de Reportes).
obtenerEstadisticas... (Líneas 23-24)	**Mover**	Extraer a LibroEstadisticaRepository.
findLibrosConPocasCopias... (Líneas 27-28)	**Eliminar**	La regla < 2 es lógica de negocio. Debe gestionarse en la capa Service.

3. Código Refactorizado (Solución Final)

A continuación, se presenta la implementación técnica corregida aplicando SOLID.

A. Nueva Interfaz: LibroRepository.java (Limpia y Flexible)

Java
```
package com.biblioteca.repository;

import com.biblioteca.model.Libro;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
import java.util.List;

@Repository
public interface LibroRepository extends JpaRepository<Libro, Long> {

    // Búsquedas básicas (Mantenidas)
    List<Libro> findByTituloContaining(String titulo);
    List<Libro> findByAutorContaining(String autor);
    List<Libro> findByIsbn(String isbn);

    // SOLUCIÓN OCP: Método genérico que reemplaza a
findLibrosReferencia/Especiales
    List<Libro> findByCategoria(String categoria);

    // SOLUCIÓN OCP/SRP: Métodos dinámicos.
    // El servicio decide los valores (0, 2, 5, etc), no el repositorio.
    List<Libro> findByCopiasDisponiblesGreaterThan(int cantidad);
    List<Libro> findByCopiasDisponiblesLessThan(int cantidad);
}
```

B. Nueva Interfaz: LibroEstadisticaRepository.java (Segregación)

```
Java
package com.biblioteca.repository;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.Repository;
import java.util.List;
import com.biblioteca.model.Libro;

@org.springframework.stereotype.Repository
public interface LibroEstadisticaRepository extends Repository<Libro, Long> {

    // Métodos de reporte aislados (Solución SRP e ISP)
    @Query("SELECT COUNT(l) FROM Libro l WHERE l.categoria = ?1")
    Long contarLibrosPorCategoria(String categoria);

    @Query("SELECT l.categoria, COUNT(l) FROM Libro l GROUP BY l.categoria")
    List<Object[]> obtenerEstadisticasPorCategoria();
}
```

4. Violación del Principio de Responsabilidad Única (SRP) y Separación de Intereses en PrestamoRepository

- **Definición:** Una clase debe tener una sola razón para cambiar. El repositorio solo debe encargarse de *acceder* a los datos, no de *interpretar* su estado de negocio.
- **Hallazgo:** El método findPrestamosVencidos contiene Lógica de Negocio Hardcodeada (Incrustada) dentro de la consulta SQL (@Query).
    - Define "Vencido" como: fecha < CURRENT_TIMESTAMP Y estado = 'ACTIVO'.
- **Por qué está mal:** El repositorio está decidiendo qué reglas constituyen un préstamo vencido.
    - Si el negocio decide dar un "período de gracia" de 3 días antes de considerarlo vencido, debes cambiar el Repositorio.
    - Si quieres buscar préstamos que *se vencerán mañana* (para enviar alertas), este método no sirve.
    - Dificulta las pruebas unitarias (Testing): No puedes controlar CURRENT_TIMESTAMP de la base de datos fácilmente desde los tests de Java.

A. Violación del Principio Abierto/Cerrado (OCP)

- **Definición:** Abierto a extensión, cerrado a modificación.
- **Hallazgo:** La consulta está cerrada a nuevos requerimientos de tiempo o estado. Al usar CURRENT_TIMESTAMP dentro de la query, el método es rígido. No se puede extender su uso para reportes históricos (ej: "¿Cuáles préstamos estaban vencidos el mes pasado?").

5. Plan de Cambios (Línea por Línea)

Se instruye realizar las siguientes modificaciones en el archivo original:

Bloque de Código (Líneas aprox.)	Acción	Justificación Técnica
findByEstado, findByUsuarioId... (Líneas 11-13)	**Mantener**	Son métodos de búsqueda estándar correctos.
@Query("SELECT p ... CURRENT_TIMESTAMP ...") (Líneas 15-16)	**Eliminar**	La lógica de tiempo (CURRENT_TIMESTAMP) y estado fijo ('ACTIVO') debe salir de la query.
findPrestamosVencidos() (Línea 17)	**Reemplazar**	Cambiar por un método genérico parametrizado: findByFechaDevolucionEsperadaBeforeAnd

		Estado.

6. Código Refactorizado (Solución Final)

A. PrestamoRepository.java (Flexible y Reutilizable)

Este repositorio ahora cumple con SRP y OCP. Puede ser usado para buscar vencidos hoy, vencidos ayer, o por vencer mañana, sin cambiar una sola línea de código aquí.

Java
```
package com.biblioteca.repository;

import com.biblioteca.model.Prestamo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.time.LocalDateTime; // Asumiendo uso de Java 8+

@Repository
public interface PrestamoRepository extends JpaRepository<Prestamo, Long> {

    List<Prestamo> findByEstado(String estado);
    List<Prestamo> findByUsuarioId(Long usuarioId);
    List<Prestamo> findByLibroId(Long libroId);

    // CORRECCIÓN SOLID:
    // En lugar de 'findPrestamosVencidos', creamos un método dinámico.
    // Spring Data JPA implementa esto automáticamente basándose en el nombre.
    // Busca préstamos donde la fecha esperada es ANTES de la fecha dada Y el estado coincide.
    List<Prestamo> findByFechaDevolucionEsperadaBeforeAndEstado(LocalDateTime fechaLimite, String estado);
}
```

7. Principios SOLID Violados UsuarioRepository

A. Single Responsibility Principle (SRP)
- El repositorio contiene reglas de negocio (líneas 16–19).

B. Open/Closed Principle (OCP)
- Uso de valores rígidos como 5 y 'VENCIDO' (líneas 18 y 21).

C. Dependency Inversion Principle (DIP)
- Falta de servicio que abstraiga reglas de negocio.

8. Cambios Necesarios (Líneas Exactas)

Línea 18 – reemplazar por consulta parametrizada:
@Query("SELECT u FROM Usuario u WHERE SIZE(u.prestamos) >= :limite")
List<Usuario> findUsuariosConLimiteAlcanzado(@Param("limite") int limite);

Línea 21 – reemplazar por:
@Query("SELECT DISTINCT u FROM Usuario u JOIN u.prestamos p WHERE p.estado = :estado")
List<Usuario> findUsuariosConPrestamosPorEstado(@Param("estado") String estado);

9. Repositorio Corregido

```
package com.biblioteca.repository;

import com.biblioteca.model.Usuario;
import com.biblioteca.model.EstadoPrestamo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    Usuario findByEmail(String email);
    List<Usuario> findByTipoUsuario(String tipoUsuario);
    List<Usuario> findByActivoTrue();

    @Query("SELECT u FROM Usuario u WHERE SIZE(u.prestamos) >= :limite")
    List<Usuario> findUsuariosConLimiteAlcanzado(@Param("limite") int limite);

    @Query("SELECT DISTINCT u FROM Usuario u JOIN u.prestamos p WHERE p.estado = :estado")
    List<Usuario> findUsuariosConPrestamosPorEstado(@Param("estado") String estado);
}
```