PATRONES DE ARQUITECTURA DE SOFTWARE

JOSE CARLOS CARRILLO VILLAZON

LUIS JOSE ARIÑO BULA

TALLER 1

PRINCIPIOS SOLID

ING. JAIRO SEOANES

UNIVERSIDAD POPULAR DEL CESAR

FACULTAD DE INGENIERIAS Y TECNOLÓGIAS

ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE

DIAGRAMA DE CASO DE USO:

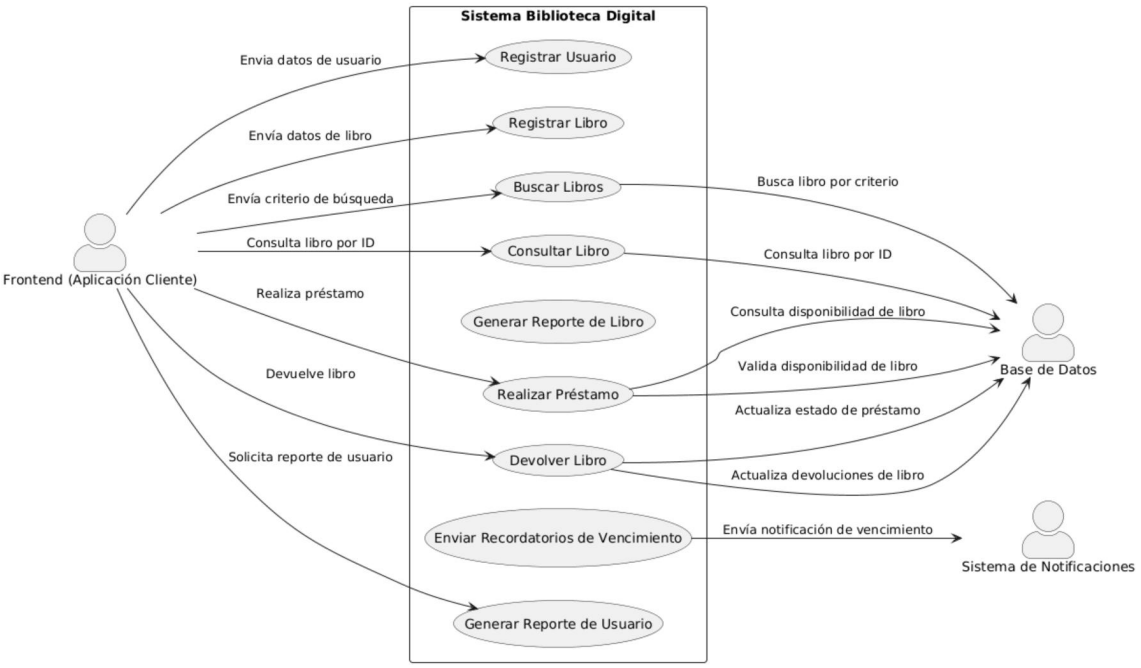
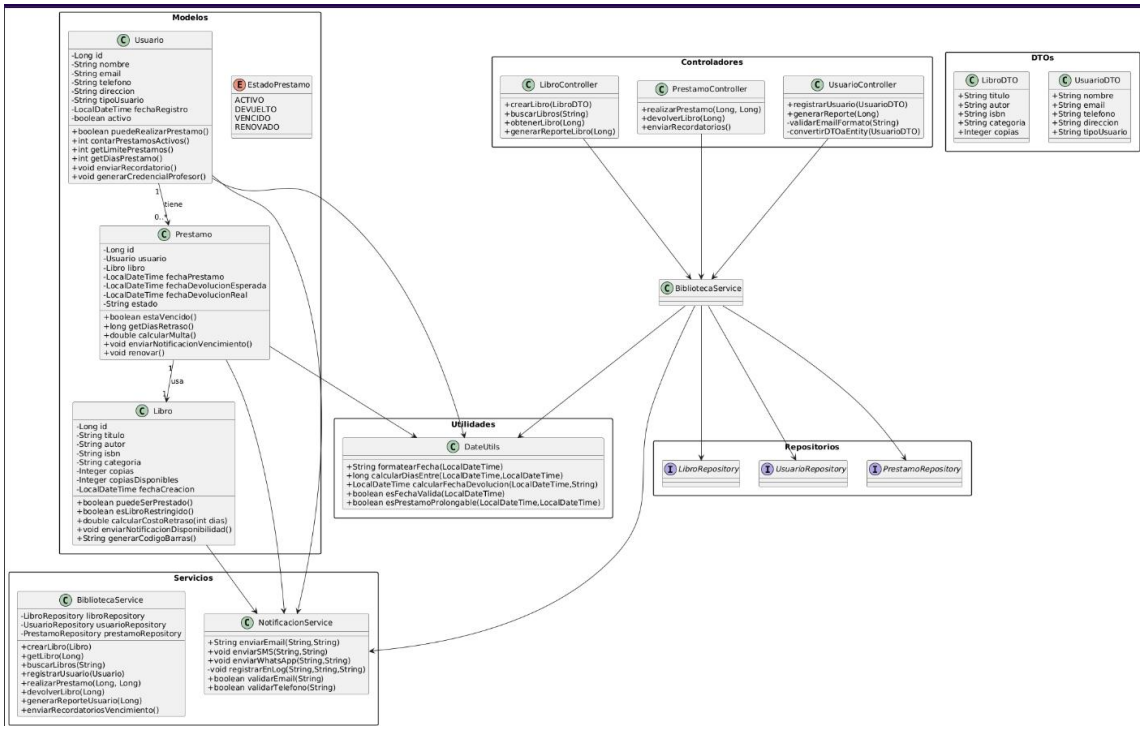
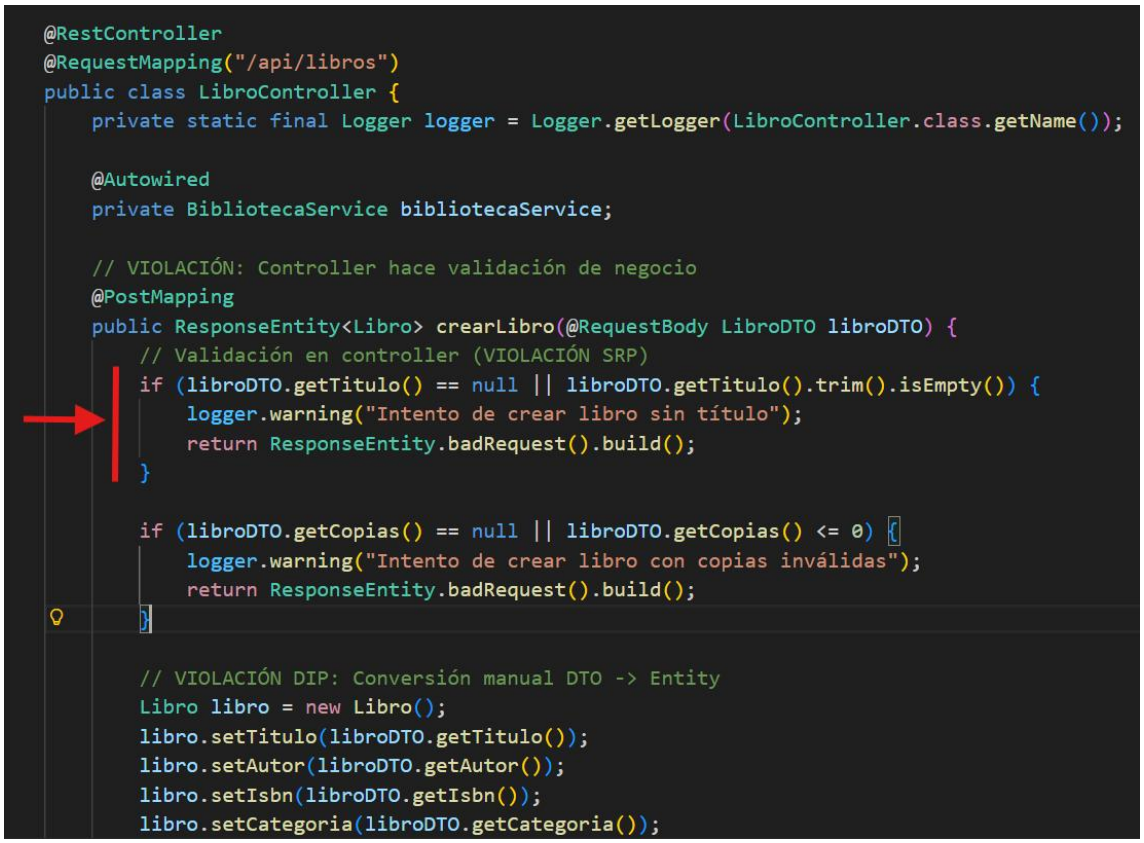


DIAGRAMA DE CLASES:



1. S — Single Responsibility Principle (SRP)

1.1 Validación de negocio en el Controller (LibroController)



```java
@RestController
@RequestMapping("/api/libros")
public class LibroController {
    private static final Logger logger = Logger.getLogger(LibroController.class.getName());

    @Autowired
    private BibliotecaService bibliotecaService;

    // VIOLACIÓN: Controller hace validación de negocio
    @PostMapping
    public ResponseEntity<Libro> crearLibro(@RequestBody LibroDTO libroDTO) {
        // Validación en controller (VIOLACIÓN SRP)
        if (libroDTO.getTitulo() == null || libroDTO.getTitulo().trim().isEmpty()) {
            logger.warning("Intento de crear libro sin título");
            return ResponseEntity.badRequest().build();
        }

        if (libroDTO.getCopias() == null || libroDTO.getCopias() <= 0) {
            logger.warning("Intento de crear libro con copias inválidas");
            return ResponseEntity.badRequest().build();
        }

        // VIOLACIÓN DIP: Conversión manual DTO -> Entity
        Libro libro = new Libro();
        libro.setTitulo(libroDTO.getTitulo());
        libro.setAutor(libroDTO.getAutor());
        libro.setIsbn(libroDTO.getIsbn());
        libro.setCategoria(libroDTO.getCategoria());
```

¿Por qué rompe SRP?

El controller está asumiendo lógica de validación → eso pertenece al servicio o a un validador.

Solución:

Mover validaciones a:

- LibroValidator
- O lógica de dominio
- O validación automática usando @Valid + Bean Validation

@PostMapping

public ResponseEntity<Libro> crearLibro(@Valid @RequestBody LibroDTO libroDTO)

{

Libro libroCreado = bibliotecaService.crearLibro(libroDTO);

return ResponseEntity.ok(libroCreado);

}

1.2 Generación de reportes dentro del Controller

```java
// VIOLACIÓN SRP: Controller genera reportes
@GetMapping("/{id}/reporte")
public ResponseEntity<String> generarReporteLibro(@PathVariable Long id) {

    try {
        // Lógica de reporte mezclada en controller // Lógica de reporte mezclada en controller
        Libro libro = this.bibliotecaService.getLibro(id);
        logger.info("Consultando libro con ID: " + libro.getTitulo());
        StringBuilder reporte = new StringBuilder();
        reporte.append("======= REPORTE DE LIBRO =======").append("<br>");
        reporte.append("ID: ").append(id).append("<br>");
        reporte.append("TITULO: ").append(libro.getTitulo()).append("<br>");
        reporte.append("AUTOR: ").append(libro.getAutor()).append("<br>");
        reporte.append("ISBN: ").append(libro.getIsbn()).append("<br>");
        reporte.append("CATEGORIA: ").append(libro.getCategoria()).append("<br>");
        reporte.append("COPIAS: ").append(libro.getCopias()).append("<br>");
        reporte.append("================================").append("<br>");
        return ResponseEntity.ok(reporte.toString());

    } catch (Exception e) {
        logger.severe("Error al obtener libro: " + e.getMessage());
        return ResponseEntity.notFound().build();
    }
```

¿Por qué rompe SRP?

El controller construye un reporte HTML, lo cual es lógica de presentación/formateo.

Solución:

Crear un servicio dedicado:

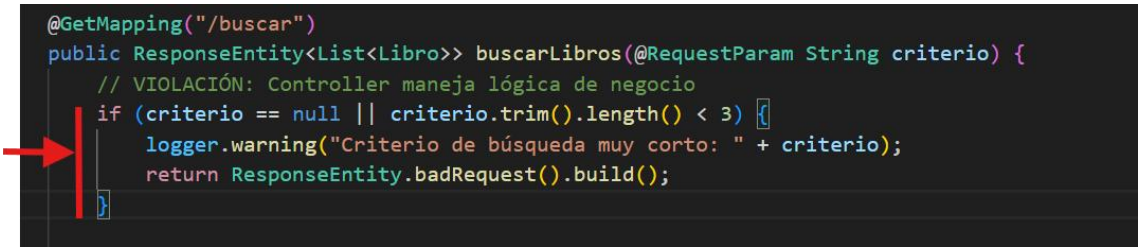
```
public interface ReporteService {
   String generarReporteLibro(Long id);
}
```

El controller solo lo llama:

```
@GetMapping("/{id}/reporte")
public ResponseEntity<String> generarReporteLibro(@PathVariable Long id)
 {
   return ResponseEntity.ok(reporteService.reporteLibro(id));
}
```

2. O — Open/Closed Principle (OCP)

2.1 Controladores devolviendo errores hardcodeados



```
@GetMapping("/buscar")
public ResponseEntity<List<Libro>> buscarLibros(@RequestParam String criterio) {
    // VIOLACIÓN: Controller maneja lógica de negocio
    if (criterio == null || criterio.trim().length() < 3) {
        logger.warning("Criterio de búsqueda muy corto: " + criterio);
        return ResponseEntity.badRequest().build();
    }
```

¿Por qué rompe OCP?

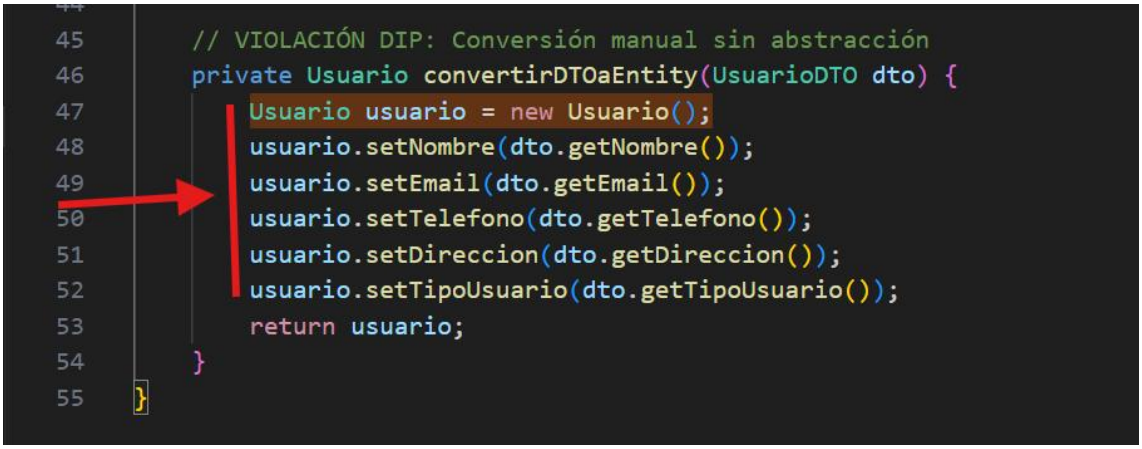
Si la regla de longitud cambia (3 → 2 → 5), el controller debe cambiar.
Eso es una violación.

Solución:

Delegar en un servicio o validador

buscarLibroValidator.validarCriterio(criterio);

2.2 Conversión manual DTO → Entity



¿Por qué rompe OCP?

Cada vez que se agregue un campo nuevo en Usuario/DTO → hay que modificar el controller.

Solución:

Crear un mapper:

```
public interface UsuarioMapper {
    Usuario toEntity(UsuarioDTO dto);
}
```

Y usarlo:

```
Usuario usuario = usuarioMapper.toEntity(dto);
```

3. L — Liskov Substitution Principle (LSP)

3.1Uso de entidades directamente en Request/Response

¿Por qué rompe LSP?

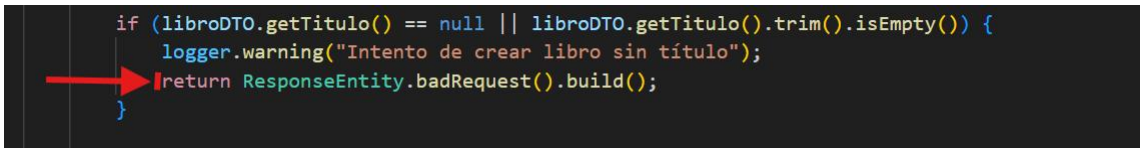
Las entidades NO cumplen contrato estable → contienen lógica, anotaciones, proxies de Hibernate, lazy-loaders.
No son sustituibles por una clase hija (DTO), lo cual es violación indirecta.

Solución:

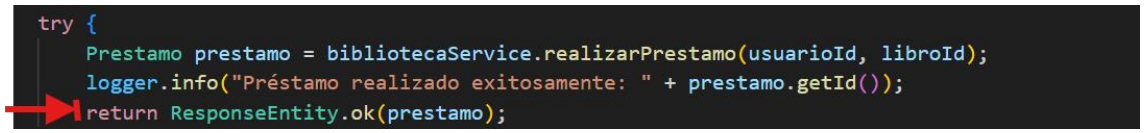
Siempre usar DTO para entrada y salida:

public ResponseEntity<LibroResponse> crearLibro(@RequestBody LibroDTO dto)

3.2 Métodos que devuelven diferentes tipos según la condición



También :



¿Por qué rompe LSP?

El controlador presenta multiplicidad de comportamientos no substituibles.
Un cliente que espera siempre datos puede fallar si recibe “null body”.

Solución:

Estandarizar respuesta:

public ResponseEntity<ResponseBase<T>> response(...)

Con estructura uniforme:

class ResponseBase<T> {

private boolean success;

private T data;

private String message;

}

4. I — Interface Segregation Principle (ISP)

4.1 Controller depende de un servicio enorme

Y esto se repite en:

- PrestamoController
- UsuarioController
- LibroController

¿Por qué rompe ISP?
Un controller que solo hace préstamos NO debería depender de métodos para gestionar usuarios, reportes, etc.

Solución:

Crear servicios específicos:

interface LibroService { ... }

interface UsuarioService { ... }

interface PrestamoService { ... }

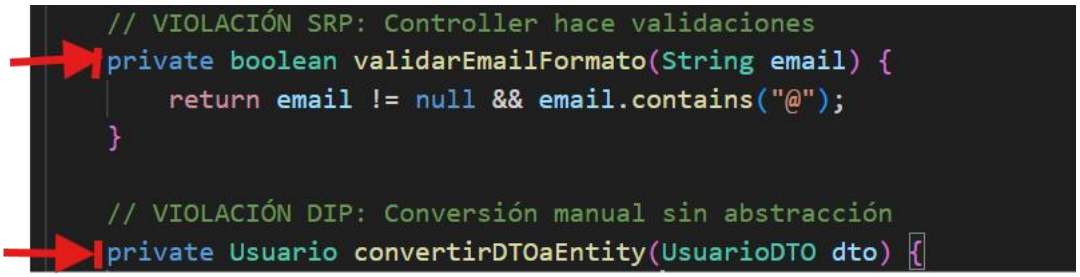
interface ReporteService { ... }

Y controllers minimalistas:

@Autowired

private PrestamoService prestamoService;

4.2 Controller de Usuario usa métodos que no son parte del "contrato" del caso de uso.



¿Por qué rompe ISP?

El controller implementa lógica que debería estar en interfaces separadas, no mezcladas.

Solución:

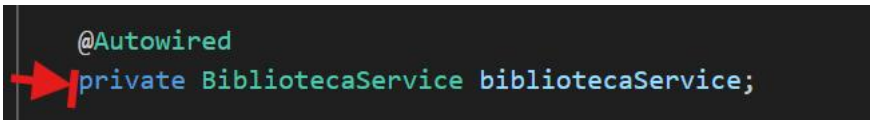
Crear interfaces pequeñas:

interface EmailValidator { boolean validar(String email); }

interface UsuarioMapper { Usuario toEntity(UsuarioDTO dto); }

5. D — Dependency Inversion Principle (DIP)

5.1 Controller depende de servicios concretos



BibliotecaService es una implementación concreta, no es una abstracción.

¿Por qué rompe DIP?

Si cambiamos la implementación, hay que tocar el controlador.

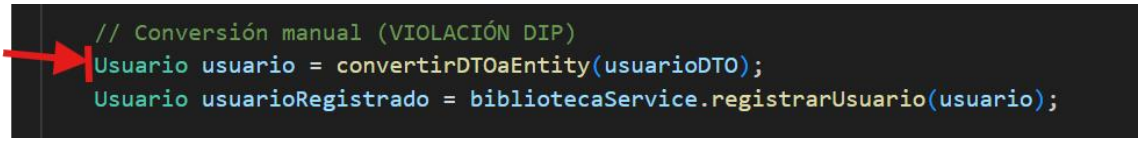
Solución:

Siempre depender de interfaces:

@Autowired

private IPrestamoService prestamoService;

5.1 Controller crea su propia lógica de mapeo/validación



El controller depende **directamente** de la entidad de dominio.

¿Por qué rompe DIP?

Debe depender de una abstracción como UsuarioMapper.

Solución:

@Autowired

private UsuarioMapper mapper;

Usuario usuario = mapper.toEntity(dto);