Instituto Politécnico Nacional
Escuela Superior de Cómputo (ESCOM)

Ingeniería de Software

Práctica 3.- Modelo de Diseño

Torres Ruiz Axel Alejandro
Boleta: 2022630178
Grupo: 7CV3

Junio de 2025

Contents

1 Resumen Ejecutivo

Visión General del Proyecto

El presente documento presenta un análisis exhaustivo del sistema **Papelio**, una plataforma de búsqueda de artículos académicos desarrollada con tecnologías modernas de Java y Spring Boot. Este análisis incluye la documentación completa de diagramas UML que cubren desde el modelo conceptual hasta la implementación y despliegue del sistema.

1.1 Objetivos del Análisis

Los objetivos principales de este análisis UML son:

1. **Modelado Conceptual:** Establecer un modelo de dominio claro que represente las entidades y relaciones del sistema académico.
2. **Análisis de Comportamiento:** Documentar los flujos de interacción principales a través de diagramas de secuencia y robustez.
3. **Diseño de Interfaces:** Modelar la experiencia del usuario y los flujos de navegación del sistema.
4. **Arquitectura de Implementación:** Definir la estructura técnica y los patrones de diseño utilizados.
5. **Estrategia de Despliegue:** Documentar las opciones de implementación tanto tradicional como containerizada.

1.2 Alcance del Sistema

El sistema Papelio abarca las siguientes funcionalidades principales:

- **Búsqueda de Artículos:** Integración con APIs académicas (Semantic Scholar, CrossRef, ArXiv)
- **Gestión de Usuarios:** Registro, autenticación y control de acceso basado en roles
- **Personalización:** Historial de búsquedas, favoritos y recomendaciones personalizadas
- **Seguridad:** Implementación de estándares GDPR y seguridad avanzada
- **Rendimiento:** Estrategias de cache y optimización para alta concurrencia
- **Administración:** Panel de control para gestión del sistema y usuarios

1.3 Tecnologías Implementadas

Categoría	Tecnología	Versión
Framework Principal	Spring Boot	3.4.2
Lenguaje	Java	21 LTS
Seguridad	Spring Security	6.x
Persistencia	Spring Data JPA	3.x
Base de Datos	PostgreSQL	15+
Cache	Caffeine	Latest
Frontend	Thymeleaf + Bootstrap	3.x + 5.x
Resiliencia	Resilience4j	2.1.0
Contenedores	Docker + Compose	Latest
Testing	JUnit 5 + Mockito	Latest
Calidad	SonarQube + JaCoCo	Latest

Table 1: Stack tecnológico del sistema Papelio

2 Análisis del Dominio

2.1 Modelo Conceptual del Dominio

El modelo conceptual del sistema Papelio representa las entidades principales del dominio académico y sus relaciones. Este modelo se basa en los siguientes conceptos fundamentales:

2.1.1 Entidades Principales

- **Usuario (User):** Entidad central que representa a los investigadores y administradores del sistema
- **Artículo (Article):** Representa las publicaciones académicas obtenidas de fuentes externas
- **Búsqueda (Search):** Encapsula las consultas realizadas por los usuarios
- **Historial (History):** Mantiene el registro de actividades del usuario
- **Favoritos (Favorites):** Gestiona las colecciones personales de artículos
- **Proveedor Externo (External Provider):** Abstrae las APIs académicas externas

2.1.2 Relaciones del Dominio

Entidad Origen	Relación	Entidad Destino	Cardinalidad
Usuario	realiza	Búsqueda	1:N
Usuario	tiene	Historial	1:N
Usuario	mantiene	Favoritos	M:N
Búsqueda	retorna	Artículo	N:M
Artículo	proviene de	Proveedor Externo	N:1
Historial	registra	Búsqueda	1:1
Historial	incluye	Visualización	1:N

Table 2: Relaciones principales del modelo de dominio

2.2 Modelo Entidad-Relación

El modelo entidad-relación traduce el modelo conceptual a una estructura de base de datos normalizada:

2.2.1 Entidades de Persistencia

1. **usuarios**
   - `id` (BIGINT, PK, AUTO_INCREMENT)
   - `email` (VARCHAR(255), UNIQUE, NOT NULL)
   - `password` (VARCHAR(255), NOT NULL)
   - `name` (VARCHAR(255), NOT NULL)
   - `role` (VARCHAR(50), DEFAULT 'USER')
   - `created_date` (TIMESTAMP)
   - `last_modified_date` (TIMESTAMP)
2. **search_history**
   - `id` (BIGINT, PK, AUTO_INCREMENT)
   - `user_email` (VARCHAR(255), FK, NOT NULL)
   - `search_query` (TEXT, NOT NULL)
   - `search_date` (TIMESTAMP, NOT NULL)
   - `results_count` (INTEGER)
3. **article_favorites**
   - `id` (BIGINT, PK, AUTO_INCREMENT)
   - `user_email` (VARCHAR(255), FK, NOT NULL)
   - `article_id` (VARCHAR(255), NOT NULL)
   - `title` (TEXT, NOT NULL)
   - `favorite_date` (TIMESTAMP)

4. **article_view_history**
    - `id` (BIGINT, PK, AUTO_INCREMENT)
    - `user_email` (VARCHAR(255), FK, NOT NULL)
    - `article_id` (VARCHAR(255), NOT NULL)
    - `title` (TEXT, NOT NULL)
    - `view_date` (TIMESTAMP)

2.2.2 Índices y Constraintes

- **Índices Únicos:** email en tabla usuarios
- **Índices Compuestos:** (user_email, search_date) en search_history
- **Índices B-Tree:** user_email en todas las tablas de historial
- **Claves Foráneas:** Todas las referencias a user_email
- **Restricciones CHECK:** Validación de formato email y roles válidos

3 Análisis de Comportamiento Dinámico

3.1 Casos de Uso Principales

El sistema Papelio implementa los siguientes casos de uso principales:

3.1.1 Caso de Uso Principal: Búsqueda Básica de Artículos

Descripción del Caso de Uso

Título: Búsqueda Básica de Artículos Académicos
Actor Principal: Usuario autenticado
Objetivo: Permitir al usuario buscar artículos académicos utilizando términos de búsqueda específicos
Precondiciones: Usuario autenticado en el sistema
Postcondiciones: Resultados de búsqueda mostrados y guardados en historial

Flujo Principal:

1. Usuario ingresa términos de búsqueda en el formulario
2. Sistema valida los parámetros de entrada
3. Sistema ejecuta búsqueda en múltiples proveedores (Strategy Pattern)
4. Sistema normaliza y consolida resultados
5. Sistema aplica filtros y ordenamiento
6. Sistema guarda la búsqueda en el historial del usuario

7. Sistema muestra resultados paginados al usuario
8. Usuario puede interactuar con los resultados (ver detalles, agregar a favoritos)

Flujos Alternativos:

- **3a.** Error en API externa: Sistema utiliza circuit breaker y fallback
- **4a.** Sin resultados: Sistema muestra sugerencias de búsqueda alternativas
- **7a.** Timeout: Sistema muestra resultados parciales con notificación

3.1.2 Casos de Uso Secundarios

Caso de Uso	Actor	Complejidad
Registro de Usuario	Usuario Anónimo	Media
Autenticación	Usuario Registrado	Baja
Gestión de Favoritos	Usuario Autenticado	Media
Visualización de Historial	Usuario Autenticado	Baja
Administración de Usuarios	Administrador	Alta
Configuración del Sistema	Administrador	Alta
Generación de Reportes	Administrador	Media

Table 3: Casos de uso secundarios del sistema

3.2 Diagramas de Secuencia

Los diagramas de secuencia documentan la interacción temporal entre los objetos del sistema para cada caso de uso.

3.2.1 Secuencia: Búsqueda Básica de Artículos

El diagrama de secuencia principal muestra las siguientes interacciones:

1. **Fase de Autenticación:**
   - Usuario → AuthController: Verificación de sesión
   - AuthController → SecurityContext: Validación de autenticación
   - SecurityContext → Usuario: Confirmación de acceso
2. **Fase de Búsqueda:**
   - Usuario → SearchController: Envío de formulario de búsqueda
   - SearchController → SearchService: Procesamiento de consulta
   - SearchService → SearchStrategyContext: Selección de estrategia
   - SearchStrategyContext → ExternalAPIClient: Llamadas a APIs
   - ExternalAPIClient → APIs Externas: Solicitudes HTTP
3. **Fase de Procesamiento:**

- APIs Externas → ExternalAPIClient: Respuestas JSON
- ExternalAPIClient → ResponseFactory: Normalización de datos
- ResponseFactory → SearchService: DTOs consolidados
- SearchService → CacheManager: Almacenamiento en cache

4. **Fase de Persistencia:**
   - SearchService → SearchHistoryService: Guardado de historial
   - SearchHistoryService → SearchHistoryRepository: Persistencia JPA
   - SearchHistoryRepository → Database: Operaciones SQL
5. **Fase de Respuesta:**
   - SearchService → SearchController: Resultados procesados
   - SearchController → ThymeleafView: Renderizado de vista
   - ThymeleafView → Usuario: Página de resultados HTML

3.2.2 Aspectos Técnicos de las Secuencias

Consideraciones de Rendimiento

- **Timeouts:** Configurados a 30 segundos para APIs externas
- **Circuit Breakers:** Implementados con Resilience4j para fallos en cascada
- **Caching:** Respuestas cacheadas por 30 minutos usando Caffeine
- **Asíncrono:** Operaciones de logging y métricas ejecutadas en segundo plano

3.3 Diagramas de Robustez

Los diagramas de robustez proporcionan una vista de análisis que conecta los casos de uso con el diseño del sistema.

3.3.1 Elementos de Robustez

- **Objetos Boundary (Frontera):** Interfaces de usuario y APIs
  - SearchForm, ResultsPage, LoginForm
  - SearchRestAPI, UserRestAPI, AdminRestAPI
- **Objetos Control (Control):** Lógica de procesamiento
  - SearchController, UserController, AdminController
  - SearchService, UserService, AuthenticationService
- **Objetos Entity (Entidad):** Datos y persistencia
  - User, SearchHistory, ArticleFavorite
  - ExternalAPIResponse, CacheEntry

3.3.2 Flujos de Robustez

El análisis de robustez valida que:

- Los actores solo interactúan con objetos boundary
- Los objetos boundary solo se comunican con objetos control
- Los objetos control acceden a objetos entity para persistencia
- No hay comunicación directa entre boundary y entity

4 Diseño de Interfaces y Navegación

4.1 Modelo de Interfaz de Usuario

El diseño de la interfaz de usuario del sistema Papelio sigue principios de usabilidad y accesibilidad modernas.

4.1.1 Arquitectura de Interfaz

- **Responsive Design:** Adaptable a dispositivos móviles y desktop
- **Framework CSS:** Bootstrap 5.x para consistencia visual
- **Template Engine:** Thymeleaf con fragmentos reutilizables
- **JavaScript:** ES6+ con jQuery para interactividad
- **Gestión de Temas:** Soporte para modo oscuro/claro/automático

4.1.2 Wireframes Principales

Página	Elementos Principales	Funcionalidad
Login	Formulario de autenticación Enlace de registro Recuperación de contraseña	Inicio de sesión Crear nueva cuenta Resetear credenciales
Dashboard	Barra de búsqueda Historial reciente Favoritos destacados Recomendaciones	Búsqueda principal Acceso rápido a búsquedas Artículos guardados Sugerencias personalizadas
Resultados	Lista de artículos Filtros y ordenamiento Paginación Acciones por artículo	Resultados de búsqueda Refinamiento de resultados Navegación entre páginas Favoritos, compartir, citar
Perfil	Información personal Configuración de cuenta Estadísticas de uso	Datos del usuario Preferencias y seguridad Métricas personales
Admin	Gestión de usuarios Métricas del sistema Configuración Logs y auditoría	CRUD de usuarios Dashboards analíticos Parámetros del sistema Monitoreo de actividad

Table 4: Wireframes y elementos de interfaz

4.2 Navegación Basada en Roles

El sistema implementa un modelo de navegación diferenciado según el rol del usuario.

4.2.1 Flujo de Navegación para Usuario Estándar

1. **Punto de Entrada:** Página de login
2. **Dashboard Principal:** Acceso tras autenticación exitosa
3. **Funcionalidades Disponibles:**
   - Búsqueda de artículos
   - Visualización de resultados
   - Gestión de favoritos
   - Historial personal
   - Configuración de perfil
4. **Restricciones:** Sin acceso a funciones administrativas

4.2.2 Flujo de Navegación para Administrador

1. **Punto de Entrada:** Página de login (credenciales admin)
2. **Dashboard Administrativo:** Panel de control extendido

3. **Funcionalidades Adicionales:**
   - Todas las funciones de usuario estándar
   - Gestión de usuarios del sistema
   - Configuración global del sistema
   - Métricas y analytics
   - Logs de auditoría
   - Gestión de APIs externas
4. **Privilegios:** Acceso completo al sistema

4.2.3 Controles de Acceso

- **Nivel de Método:** Anotaciones `@PreAuthorize` en controladores
- **Nivel de Vista:** Condicionales Thymeleaf basadas en roles
- **Nivel de URL:** Configuración de Spring Security
- **Nivel de API:** Validación en REST endpoints

5 Arquitectura de Implementación

5.1 Patrones de Diseño Implementados

El sistema Papelio implementa varios patrones de diseño para asegurar mantenibilidad, extensibilidad y robustez.

5.1.1 Patrón Strategy - Motor de Búsqueda Multi-Proveedor

Implementación del Patrón Strategy

Problema: Necesidad de integrar múltiples APIs académicas con diferentes interfaces y comportamientos.
Solución: Implementación del patrón Strategy para encapsular algoritmos de búsqueda específicos por proveedor.

Componentes del Patrón:

1. **Contexto (SearchEngineContext):**

```
@Service
public class SearchEngineContext {
    private final List<SearchStrategy> strategies;
    private final SearchStrategySelector selector;

    public SearchResponseDTO executeSearch(SearchRequestDTO request
        ) {
        SearchStrategy strategy = selector.selectStrategy(request);
        return strategy.search(request);
    }
}
```

2. **Estrategia Abstracta (SearchStrategy):**

```
public interface SearchStrategy {
    SearchResponseDTO search(SearchRequestDTO request);
    boolean supports(SearchRequestDTO request);
    String getProviderName();
    int getPriority();
}
```

3. **Estrategias Concretas:**

   - `SemanticScholarStrategy`: Búsqueda en Semantic Scholar API
   - `CrossRefStrategy`: Búsqueda en CrossRef API
   - `ArXivStrategy`: Búsqueda en ArXiv API
   - `FallbackStrategy`: Estrategia de respaldo para fallos

Beneficios de la Implementación:

- **Extensibilidad:** Fácil adición de nuevos proveedores
- **Mantenibilidad:** Lógica específica encapsulada por proveedor
- **Testabilidad:** Cada estrategia puede probarse independientemente
- **Flexibilidad:** Selección dinámica de estrategia en tiempo de ejecución

5.1.2 Patrón Factory - Creación de Respuestas y DTOs

Implementación del Patrón Factory

Problema: Necesidad de crear objetos de respuesta normalizados a partir de diferentes fuentes de datos.
Solución: Implementación del patrón Factory para centralizar la creación de objetos de transferencia de datos.

Componentes del Patrón:

1. **Factory Abstract (ResponseFactory):**

```
public abstract class ResponseFactory {
    public abstract SearchResponseDTO createSearchResponse(
        ExternalAPIResponse apiResponse,
        SearchContext context
    );

    public abstract ErrorResponseDTO createErrorResponse(
        Exception exception,
        SearchContext context
    );
}
```

2. **Factories Concretas:**

   - `SemanticScholarResponseFactory`

- `CrossRefResponseFactory`
- `ArXivResponseFactory`
- `GenericErrorResponseFactory`

3. **Factory Registry:**

```
@Component
public class FactoryRegistry {
    private final Map<String, ResponseFactory> factories;

    public ResponseFactory getFactory(String providerName) {
        return factories.get(providerName);
    }
}
```

5.1.3 Patrón Repository - Acceso a Datos

- **Interfaces de Repositorio:** Abstracción del acceso a datos
- **Implementación JPA:** Spring Data JPA con consultas personalizadas
- **Transaccionalidad:** Gestión automática de transacciones
- **Caching:** Integración con el sistema de cache

5.1.4 Patrón MVC - Arquitectura Web

- **Model:** Entidades JPA y DTOs
- **View:** Plantillas Thymeleaf con fragmentos
- **Controller:** Controladores Spring MVC y REST
- **Separación:** Clara separación de responsabilidades

5.2 Arquitectura Técnica por Capas

5.2.1 Capa de Presentación

- **Web Controllers:** Manejo de peticiones HTTP y renderizado de vistas
- **REST Controllers:** APIs JSON para AJAX y integraciones
- **Templates:** Vistas Thymeleaf con componentes reutilizables
- **Static Resources:** CSS, JavaScript, imágenes optimizadas

5.2.2 Capa de Lógica de Negocio

- **Services:** Lógica de dominio y orquestación de procesos
- **Strategy Implementations:** Algoritmos de búsqueda específicos
- **Integration Layer:** Comunicación con APIs externas
- **Validation:** Validación de datos de entrada y salida

5.2.3 Capa de Acceso a Datos

- **JPA Repositories:** Acceso a base de datos relacional
- **Cache Repositories:** Gestión de cache distribuido
- **External API Clients:** Clientes para servicios externos
- **Data Mappers:** Conversión entre entidades y DTOs

5.2.4 Capa de Infraestructura

- **Security:** Autenticación, autorización y protección
- **Configuration:** Gestión de configuraciones por ambiente
- **Monitoring:** Métricas, logging y health checks
- **Caching:** Estrategias de cache multinivel

6 Estrategias de Despliegue

6.1 Despliegue Tradicional Multi-Servidor

6.1.1 Arquitectura de Producción

Componente	Servidor	Especificaciones	Función
Load Balancer	Nginx	2 vCPUs 4 GB RAM 20 GB SSD	Balanceo de carga Terminación SSL Caché estático
App Server 1	Spring Boot	4 vCPUs 8 GB RAM 50 GB SSD	Aplicación principal Procesamiento Logging
App Server 2	Spring Boot	4 vCPUs 8 GB RAM 50 GB SSD	Aplicación principal Alta disponibilidad Escalabilidad
Database	PostgreSQL	8 vCPUs 16 GB RAM 500 GB SSD	Datos persistentes Transacciones Backups
Cache	Redis	4 vCPUs 8 GB RAM 100 GB SSD	Sesiones Cache de API Rate limiting
Monitoring	ELK Stack	2 vCPUs 4 GB RAM 200 GB SSD	Logs centralizados Métricas Alertas

Table 5: Arquitectura de despliegue tradicional

6.1.2 Configuración de Red y Seguridad

- **DMZ (Zona Desmilitarizada):** Load balancer expuesto públicamente

- **Red Interna:** Servidores de aplicación en red privada
- **Red de Datos:** Base de datos en subred aislada
- **Firewall:** Reglas restrictivas por puerto y protocolo
- **VPN:** Acceso administrativo seguro
- **SSL/TLS:** Certificados válidos para todos los endpoints

6.2 Despliegue Containerizado con Docker

6.2.1 Arquitectura de Contenedores

Estrategia de Containerización

El despliegue con Docker proporciona consistencia entre ambientes, facilita el escalamiento y simplifica la gestión de dependencias.

Configuración Docker Compose:

```
version: '3.8'
services:
  papelio-app:
    build: .
    ports:
      - "8080:8080"
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - DB_HOST=postgresql
      - REDIS_HOST=redis
    depends_on:
      - postgresql
      - redis
    volumes:
      - ./logs:/app/logs
      - ./config:/app/config
    networks:
      - papelio-network

  postgresql:
    image: postgres:15-alpine
    ports:
      - "5433:5432"
    environment:
      - POSTGRES_DB=papelio
      - POSTGRES_USER=papelio_user
      - POSTGRES_PASSWORD=secure_password
    volumes:
      - postgres-data:/var/lib/postgresql/data
    networks:
      - papelio-network

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
```

```
    volumes:
      - redis-data:/data
    networks:
      - papelio-network

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - papelio-app
    networks:
      - papelio-network

volumes:
  postgres-data:
  redis-data:

networks:
  papelio-network:
    driver: bridge
```

6.2.2 Dockerfile Multi-Etapa

```
# Build stage
FROM openjdk:21-jdk as builder
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN ./mvnw clean package -DskipTests

# Runtime stage
FROM openjdk:21-jre-slim
WORKDIR /app
COPY --from=builder /app/target/*.jar app.jar
EXPOSE 8080
HEALTHCHECK --interval=30s --timeout=10s --start-period=60s \
  CMD curl -f http://localhost:8080/actuator/health || exit 1
ENTRYPOINT ["java", "-jar", "app.jar"]
```

6.2.3 Beneficios del Despliegue Containerizado

- **Portabilidad:** Mismo comportamiento en todos los ambientes
- **Escalabilidad:** Fácil escalamiento horizontal
- **Aislamiento:** Dependencias encapsuladas por servicio
- **Versionado:** Control de versiones de imágenes
- **Rollback:** Reversión rápida en caso de problemas

- **Recursos:** Uso eficiente de recursos del sistema

6.3 Ambiente de Desarrollo

6.3.1 Configuración Local

- **Base de Datos:** H2 en memoria para desarrollo rápido
- **Hot Reload:** Spring Boot DevTools para cambios en tiempo real
- **Debug:** Puerto 5005 habilitado para depuración remota
- **Profiles:** Configuración específica para desarrollo
- **Docker:** Contenedor de desarrollo con volúmenes montados

6.3.2 Pipeline CI/CD

1. **Source Control:** Git con GitFlow branching strategy
2. **Build:** Maven compile y package
3. **Testing:** JUnit + Mockito + Testcontainers
4. **Quality:** SonarQube analysis + JaCoCo coverage
5. **Security:** Dependency check y vulnerability scanning
6. **Containerization:** Docker build y push to registry
7. **Deployment:** Automated deployment to staging/production

7 Validación y Consistencia

7.1 Matriz de Trazabilidad

Requisito	Dominio	Secuencia	Diseño	Arquitectura	Despliegue
Búsqueda de Artículos					
Gestión de Usuarios					
Historial y Favoritos					
Seguridad GDPR					
APIs Externas					
Caching y Performance					
Administración					
Monitoreo					

Table 6: Matriz de trazabilidad requisitos-diagramas

7.2 Validación Tecnológica

Verificación del Stack Tecnológico

Todos los diagramas han sido validados contra el archivo `pom.xml` del proyecto, asegurando que las tecnologías representadas coinciden exactamente con las dependencias declaradas.

Tecnología	Versión en pom.xml	Representada
Spring Boot	3.4.2	
Java	21	
Spring Security	6.x	
Spring Data JPA	3.x	
PostgreSQL	Driver Latest	
Thymeleaf	3.x + Security	
Spring WebFlux	3.x	
Caffeine	Latest	
Resilience4j	2.1.0	
Jackson	Latest	
JUnit 5	Latest	
JaCoCo	0.8.11	
SonarQube	Plugin 3.11.0	

Table 7: Validación tecnológica contra pom.xml

7.3 Consistencia Entre Diagramas

7.3.1 Validación de Entidades

- **Modelo de Dominio ERD:** Todas las entidades conceptuales tienen correspondencia en el modelo de datos
- **ERD Diseño de Clases:** Cada tabla de base de datos tiene su entidad JPA correspondiente
- **Secuencia Diseño:** Todos los objetos participantes en secuencias están definidos en el diagrama de clases

7.3.2 Validación de Operaciones

- **Casos de Uso Secuencia:** Cada caso de uso tiene su diagrama de secuencia correspondiente
- **Secuencia Métodos:** Todas las operaciones en secuencias están definidas como métodos en las clases
- **Interfaz Controladores:** Cada página/endpoint tiene su controlador correspondiente

7.3.3 Validación Arquitectónica

- **Capas Paquetes:** La estructura por capas se refleja en la organización de paquetes
- **Patrones Implementación:** Los patrones de diseño están correctamente implementados
- **Tecnologías Componentes:** Cada tecnología está representada en los componentes arquitectónicos

8 Métricas y Calidad

8.1 Métricas de Diseño

Métrica	Valor	Evaluación
Número de Clases	45+	Adecuado para la complejidad
Profundidad de Herencia	2-3 niveles	Óptimo
Acoplamiento	Bajo	Buena separación de responsabilidades
Cohesión	Alta	Clases con responsabilidades bien definidas
Cobertura de Casos de Uso	100%	Todos los casos cubiertos
Patrones Implementados	4+	Strategy, Factory, Repository, MVC
Capas Arquitectónicas	4	Presentación, Negocio, Datos, Infraestructura

Table 8: Métricas de calidad del diseño

8.2 Atributos de Calidad

8.2.1 Mantenibilidad

- **Modularidad:** Arquitectura por capas con responsabilidades claras
- **Reusabilidad:** Componentes reutilizables y patrones estándar
- **Testabilidad:** Dependencias inyectadas y mocks disponibles
- **Documentación:** Diagramas UML completos y documentación técnica

8.2.2 Escalabilidad

- **Horizontal:** Diseño stateless permite múltiples instancias
- **Vertical:** Arquitectura optimizada para mayor hardware
- **Caching:** Múltiples niveles de cache para reducir carga
- **Asíncrono:** Operaciones no críticas ejecutadas en background

8.2.3 Seguridad

- **Autenticación:** Múltiples mecanismos de autenticación
- **Autorización:** Control de acceso basado en roles
- **Confidencialidad:** Encriptación de datos sensibles
- **Integridad:** Validación de entrada y salida de datos

8.2.4 Rendimiento

- **Respuesta:** Tiempos de respuesta optimizados
- **Throughput:** Capacidad de procesamiento concurrente
- **Recursos:** Uso eficiente de memoria y CPU
- **Latencia:** Minimización de delays en operaciones críticas

9 Conclusiones y Recomendaciones

9.1 Logros del Análisis

Objetivos Alcanzados

- **Cobertura Completa:** Todos los aspectos del sistema han sido modelados
- **Consistencia:** Diagramas alineados entre sí y con la implementación
- **Detalle Técnico:** Nivel de detalle apropiado para implementación
- **Estándares UML:** Notación correcta y consistente
- **Documentación:** Explicaciones claras y completas

9.2 Fortalezas del Diseño

1. **Arquitectura Sólida:** Separación clara de responsabilidades por capas
2. **Patrones Apropiados:** Uso correcto de patrones de diseño reconocidos
3. **Extensibilidad:** Fácil adición de nuevas funcionalidades
4. **Mantenibilidad:** Código bien estructurado y documentado
5. **Escalabilidad:** Diseño preparado para crecimiento
6. **Seguridad:** Implementación robusta de controles de seguridad

9.3 Recomendaciones para Mejoras Futuras

9.3.1 Mejoras Técnicas

- **Microservicios:** Considerar migración a arquitectura de microservicios
- **Event Sourcing:** Implementar para mejor auditoría y trazabilidad
- **GraphQL:** Evaluar para APIs más flexibles
- **Kubernetes:** Orquestación de contenedores para producción
- **Observabilidad:** Implementar distributed tracing

9.3.2 Mejoras Funcionales

- **Machine Learning:** Recomendaciones más inteligentes
- **Colaboración:** Funciones de compartir y colaborar
- **Analytics:** Dashboards más avanzados para usuarios
- **API Pública:** Exponer APIs para integraciones externas
- **Mobile App:** Aplicación móvil nativa

9.3.3 Mejoras de Proceso

- **CI/CD:** Pipeline más robusto con múltiples ambientes
- **Monitoring:** Alertas proactivas y dashboards en tiempo real
- **Testing:** Mayor cobertura de pruebas de integración
- **Documentation:** Documentación API automatizada
- **Performance:** Pruebas de carga regulares

9.4 Impacto y Valor del Análisis

Valor Agregado del Análisis UML

Este análisis UML proporciona una base sólida para:

- **Desarrollo:** Guía clara para implementación
- **Mantenimiento:** Documentación para soporte continuo
- **Evolución:** Roadmap para futuras mejoras
- **Comunicación:** Lenguaje común para stakeholders
- **Calidad:** Fundamentos para un producto robusto

10 Referencias y Anexos

10.1 Referencias Bibliográficas

1. Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional.

2. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

3. Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.

4. Spring Team. (2024). *Spring Boot Reference Documentation*. Version 3.4.2. https://docs.spring.io boot/

5. PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*. Version 15. https://www.postgresql.org/docs/

6. Semantic Scholar. (2024). *Semantic Scholar API Documentation*. https://api.semanticscholar.org

7. CrossRef. (2024). *CrossRef REST API Documentation*. https://api.crossref.org/