

Informe de Proyecto: Migración de Sistema Monolítico a Microservicios

**Biblioteca Monolitica**

• Benjamín Palma, Gustavo Espinoza, Benjamín Leal

• Desarrollo Full Stack I

• Victor Isidro Rosendo Lugo

• Fecha de entrega 21/03/2925

**1. Definición del Problema**

1.1Descripción del sistema monolítico actual (contexto)

La biblioteca de la universidad ha estado utilizando un sistema de gestión basado en una arquitectura monolítica para administrar sus operaciones, incluyendo el catálogo de libros, préstamos, devoluciones y gestión de usuarios. Sin embargo, este enfoque ha comenzado a mostrar serias deficiencias a medida que la demanda y el volumen de datos han crecido.

1.2 Identificación del problema principal

El problema principal presenta **serias limitaciones de escalabilidad y mantenimiento**, lo que dificulta su capacidad para manejar el creciente volumen de datos y la demanda de usuarios. Esto ha generado problemas de rendimiento, dificultando la eficiencia en la administración de préstamos, devoluciones y consultas del catálogo. Además, la arquitectura actual impide una actualización modular y flexible, afectando la capacidad de adaptación a nuevas necesidades tecnológicas.

1.3 Impacto de dicho problema en el negocio/proyecto

Las deficiencias del sistema monolítico han generado varios efectos negativos en la operación de la biblioteca, afectando tanto la eficiencia del servicio como la experiencia de los usuarios:

● **Dificultad para escalar**: El sistema actual no puede manejar eficazmente un aumento en el número de usuarios concurrentes, lo que provoca tiempos de respuesta lentos y caídas del sistema.

● **Mantenimiento complicado**: Cualquier cambio en una funcionalidad del sistema requiere modificar y desplegar toda la aplicación, lo que genera tiempos de inactividad y riesgos de errores en otras partes del código.

● **Búsquedas ineficientes**: La base de datos centralizada y el código monolítico hacen que las consultas sean lentas, afectando la experiencia de los estudiantes y el personal.

● **Procesos de préstamo y devolución congestionados**: Las largas colas persisten debido a la lentitud del sistema para actualizar registros y procesar transacciones.

● **Falta de integración con nuevas tecnologías**: La biblioteca quiere implementar nuevas funciones como notificaciones automáticas, acceso remoto y analítica de uso, pero el sistema actual no lo permite fácilmente.

1.4 Objetivo general y objetivos específicos del proyecto.

### Objetivo General

Modernizar el sistema de gestión bibliotecaria mediante una arquitectura escalable y modular, mejorando el rendimiento, la experiencia del usuario y la eficiencia operativa.

### 

### Objetivos Específicos

1. **Optimizar el rendimiento del sistema** para reducir tiempos de respuesta en consultas, préstamos y devoluciones.
2. **Implementar una arquitectura escalable** que permita la expansión y adaptación a futuras necesidades sin afectar la operatividad.
3. **Mejorar la experiencia del usuario** mediante una interfaz más rápida, intuitiva y accesible.
4. **Reducir los costos de mantenimiento** facilitando la actualización y corrección de errores sin afectar a todo el sistema.
5. **Garantizar la disponibilidad y estabilidad** del servicio minimizando fallos críticos y tiempos de inactividad.

**2. Estrategia de Microservicios**

2.2 Tipo de estrategia usada (por dominio, funcionalidad, base de datos, etc.)

Para modernizar el sistema de la biblioteca, se utilizará una estrategia de **microservicios por dominio**. Esto implica dividir el sistema en servicios independientes según las principales áreas de negocio o funcionalidades, permitiendo una mejor organización y escalabilidad. Los microservicios clave serán:

* **Gestión de Usuarios**: Manejo de autenticación, registro y roles de usuario (estudiantes, docentes, bibliotecarios, administradores, etc.).
* **Catálogo de Libros**: Administración de libros, categorías, disponibilidad y búsqueda.
* **Préstamos y Devoluciones**: Control de transacciones de libros, fechas de entrega y sanciones.
* **Notificaciones**: Envio de alertas sobre devoluciones, reservas, novedades y comunicados.
* **Administración de Contenido**: Gestión de documentos digitales, comentarios, eventos y anuncios.
* **Logística y Stock**: Control de inventario y adquisiciones .

Cada microservicio operará de manera independiente y se comunicará con los demás mediante APIs, lo que garantiza flexibilidad y modularidad en el sistema.

2.2 Beneficios de adoptar esta estrategia en comparación al sistema actual

| Característica | Sistema Monolítico | Microservicio |
| --- | --- | --- |
| Escalabilidad | Limitada, afecta a todo el sistema | Independiente por servicio |
| Mantenimiento | Requiere modificar toda la aplicación | Se actualizan módulos específicos |
| Rendimiento | Se ralentiza con el crecimiento de datos | Distribución de carga eficiente |
| Disponibilidad | Un fallo afecta a toda la plataforma | Fallos localizados sin afectar el sistema completo |
| Flexibilidad | Difícil de adaptar a nuevas tecnologías | Se pueden agregar o modificar servicios fácilmente |
| Tiempo de Desarrollo | Cambios requieren mucho tiempo y pruebas globales | Desarrollo ágil con despliegue continuo |

2.3 Justificación de técnica y de diseño de la estrategia elegida

#### Descripción Estructural de la Arquitectura

La arquitectura de microservicios propuesta para el sistema de biblioteca representa un enfoque moderno y eficiente para el desarrollo de sistemas de información, caracterizado por los siguientes componentes estratégicos:

##### 1. Descomposición por Dominio Funcional

La arquitectura se compone de seis microservicios especializados, cada uno con responsabilidades específicas:

1. **Servicio de Gestión de Usuarios**
   * Centraliza la administración de perfiles
   * Gestiona autenticación y control de accesos
   * Mantiene información de usuarios independiente
2. **Servicio de Catálogo de Libros**
   * Administra el inventario bibliográfico
   * Permite búsquedas y clasificaciones
   * Mantiene actualizada la disponibilidad de recursos
3. **Servicio de Logística y Stock**
   * Controla existencias físicas
   * Gestiona adquisiciones y movimientos
   * Proporciona trazabilidad de recursos
4. **Servicio de Administración de Contenidos**
   * Gestiona recursos digitales
   * Administrar repositorios de información
   * Permite publicación de contenidos especializados
5. **Servicio de Préstamos y Devoluciones**
   * Controla transacciones de libros
   * Gestiona plazos y sanciones
   * Administra el ciclo de préstamo
6. **Servicio de Notificaciones**
   * Gestiona comunicaciones
   * Envía alertas automatizadas
   * Mantiene informados a los usuarios

**3. Herramientas Utilizadas**

3.1 Listado de herramientas (lenguajes, frameworks, bases de datos, plataformas cloud, etc.)

#### Lenguajes de Programación

* **Java**: Para el desarrollo de los microservicios backend debido a su robustez, seguridad y compatibilidad con arquitecturas distribuidas.
* **Python**: Para la implementación de módulos de análisis de datos y machine learning en la biblioteca.
* **JavaScript (Node.js)**: Para algunos microservicios livianos y la API Gateway, dado su rendimiento asíncrono y su facilidad de escalabilidad.

#### Frameworks y Librerías

* **Spring Boot (Java)**: Para la implementación de los microservicios, ya que facilita el desarrollo rápido y la integración con bases de datos y herramientas de mensajería.
* **Express.js (Node.js)**: Para la API Gateway, permitiendo manejar las peticiones de manera eficiente.
* **React.js**: Para la interfaz de usuario, asegurando una experiencia fluida y moderna.

#### Bases de Datos

* **PostgreSQL**: Base de datos relacional utilizada para almacenar la información estructurada, como usuarios, libros y préstamos.
* **MongoDB**: Base de datos NoSQL utilizada para almacenar registros de actividad, logs del sistema y notificaciones.

#### Plataformas Cloud y Contenedores

* **Docker**: Para la contenedorización de los microservicios y asegurar que funcionen de manera aislada.
* **Kubernetes**: Para la orquestación de contenedores y escalabilidad del sistema.
* **AWS (Amazon Web Services) / Google Cloud Platform**: Infraestructura en la nube para el despliegue del sistema y almacenamiento escalable.

#### Herramientas de Comunicación entre Microservicios

* **RabbitMQ o Kafka**: Para la mensajería entre microservicios y mejorar el procesamiento asincrónico.

#### Seguridad y Autenticación

* **Keycloak**: Para la gestión de autenticación y autorización basada en OAuth2 y OpenID Connect.
* **JWT (JSON Web Token)**: Para la autenticación segura de usuarios.

#### Monitoreo y DevOps

* **Prometheus y Grafana**: Para monitoreo del rendimiento del sistema y visualización de métricas.
* **Jenkins o GitHub Actions**: Para integración y despliegue continuo (CI/CD).

3.2 Funcionalidad específica de cada herramienta

| Herramienta | Funcionalidad |
| --- | --- |
| Java | Backend robusto para la gestión de usuarios, préstamos y libros. |
| Python | Análisis de datos y machine learning para sugerencias de libros y predicción. |
| Node.js | API Gateway para la comunicación entre servicios. |
| Spring Boot | Desarrollo de microservicios eficientes en Java. |
| Express.js | Creación de rutas y API Gateways en Node.js |
| React.js | Desarrollo de la interfaz de usuario moderna y responsiva. |
| PostgreSQL | Base de datos relacional para la gestión estructurada de información. |
| MongoDB | Base de datos NoSQL para registros de actividad y logs. |
| Docker | Contenerización de microservicios para despliegue eficiente. |
| Kubernetes | Orquestación y escalado de contenedores. |
| AWS | Infraestructura en la nube para alta disponibilidad y escalabilidad. |
| Kafka | Mensajería asincrónica entre microservicios. |
| Keycloak | Gestión centralizada de autenticación y permisos. |
| JWT | Token de autenticación segura para los usuarios. |
| Grafana | Monitoreo del sistema y visualización de métricas. |
| Github Actions | Integración y despliegue continuo para automatización del desarrollo |

3.3 Justificación de la elección de estas herramientas

* Java (Spring Boot) para el backend por su robustez.
* Python para análisis de datos y machine learning.
* Node.js (Express.js) para la API Gateway y microservicios ligeros.
* React.js en el frontend para una interfaz moderna.
* PostgreSQL para datos estructurados y MongoDB para logs y notificaciones.
* Docker y Kubernetes para despliegue escalable en la nube (AWS/GCP).
* RabbitMQ/Kafka para comunicación eficiente entre microservicios.
* Keycloak y JWT para autenticación segura.
* Prometheus, Grafana y CI/CD (Jenkins/GitHub Actions) para monitoreo y automatización.

**4. Herramientas de Trabajo Colaborativo**

4.1 Herramientas utilizadas (Miro, Trello, GitHub Projects, Discord, etc.)

Para la organización y seguimiento del proyecto, se utilizaron:

* **Trello**: Para gestionar tareas con tableros Kanban, dividiendo el trabajo en **Pendiente, En progreso y Completado**.
* **GitHub**: Para el control de versiones, gestión de código y seguimiento de incidencias mediante **issues y pull requests**.
* **Discord**: Para la comunicación en tiempo real, facilitando reuniones rápidas y coordinación del equipo.

4.2 Ejemplos concretos de cómo se planificó y organizó el trabajo

Para estructurar y gestionar el trabajo de manera eficiente, se implementaron las siguientes estrategias:

* **Discord**: Se realizaron **reuniones diarias por llamada grupal** para coordinar avances y resolver dudas en tiempo real.
* **Trello**: Se diseñó un flujo de trabajo claro con cuatro columnas:  
  + **Por hacer**: Tareas pendientes.
  + **En proceso**: Tareas en desarrollo.
  + **En validación**: Funcionalidades en revisión antes de implementarse.
  + **Terminado**: Tareas completadas y aprobadas.
* **GitHub**:  
  + Se creó un **repositorio** para almacenar y actualizar el documento del informe de manera frecuente.

4.3 Capturas de pantalla o enlaces a tableros como evidencia

enlace a github y trello

* **Trello**

<https://trello.com/b/ynE5U5Kk>

* **Github**

<https://github.com/benjazJJ/Proyecto-Semestral-Biblioteca>

**5. Enfoque Ético en el Desarrollo**

5.1Privacidad de los datos

## Desafío:

El software maneja datos personales de los usuarios, como nombres, direcciones, historial de préstamos y preferencias de lectura

### Soluciones y buenas prácticas:

* Aplicar **cifrado** en la base de datos para proteger la información
* Implementar **autenticación segura** (como 2FA) para el acceso
* Cumplir con normativas de protección de datos (GDPR, CCPA o equivalentes locales)
* Minimizar la recopilación de datos y permitir a los usuarios gestionar su privacidad

## 5.2 Seguridad

### Desafío:

El software podría ser vulnerable a ataques como robo de datos o accesos no autorizados.

### Soluciones y buenas prácticas:

* Usar **protocolos de seguridad** como HTTPS, cifrado AES para datos sensibles y control de acceso por roles.
* Mantener **actualizaciones regulares** para corregir vulnerabilidades.
* Realizar auditorías y pruebas de seguridad constantes.
* Implementar **backups automáticos** para evitar la pérdida de información.

5.3 Responsabilidad en el despliegue

### Desafío:

El software debe funcionar correctamente sin causar perjuicios a la biblioteca ni a sus usuarios.

### Soluciones y buenas prácticas:

* Realizar **pruebas exhaustivas** antes de la implementación.
* Asegurar que la interfaz sea **accesible** para personas con discapacidades.
* Evitar **sesgos en algoritmos** de recomendación para que no limiten el acceso a ciertos contenidos.
* Crear un sistema de **soporte técnico** para resolver problemas rápidamente.

5.4 Impacto en los puestos de trabajo

### Desafío:

Automatizar procesos podría reducir la necesidad de personal bibliotecario.

### Soluciones y buenas prácticas:

* Diseñar el software para **complementar** el trabajo humano, no reemplazarlo.
* Proveer **capacitación** a los empleados para que utilicen el sistema de manera efectiva.
* Incluir funciones que mejoren la eficiencia del personal sin eliminar su rol esencial.

5.5 Consideraciones de cumplimiento normativo (ej. Ley de Protección de Datos en Chile)

Para evaluar los principales enfoques éticos, ocuparemos las siguientes tres leyes: la **Ley 21.469**, la **Ley 21.663** y la **Ley 21.719**, las cuales abordan temas clave como la privacidad de los datos, la seguridad, la responsabilidad en el despliegue tecnológico y el impacto en los puestos de trabajo.

La **Ley 21.719**, promulgada en diciembre de 2024, establece un marco sólido para la protección de datos personales en Chile. Esta norma crea la Agencia de Protección de Datos Personales y garantiza derechos esenciales como el acceso, rectificación y supresión de información. Su implementación es fundamental para reforzar la privacidad y alinear la legislación chilena con estándares internacionales como el GDPR.

Por su parte, la **Ley 21.663**, conocida como la Ley Marco de Ciberseguridad, establece un modelo de gobernanza en la gestión de riesgos digitales. A través de la creación de la Agencia Nacional de Ciberseguridad (ANCI) y el Equipo Nacional de Respuesta a Incidentes de Seguridad Informática (CSIRT), esta normativa busca fortalecer la prevención y respuesta ante ciberataques, imponiendo obligaciones específicas a entidades públicas y privadas.

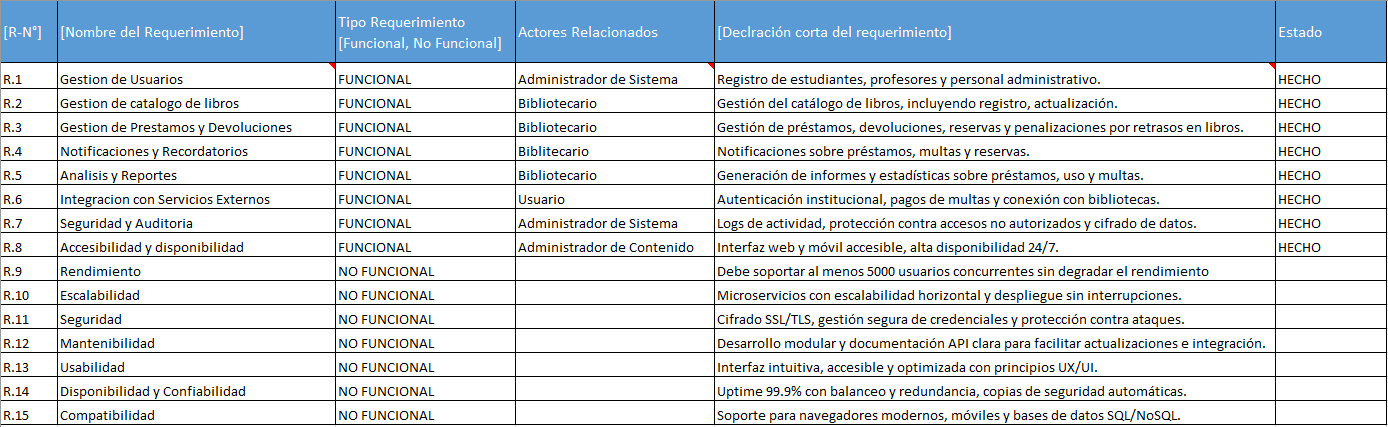
Finalmente, la **Ley 21.469** aborda el impacto de la transformación digital en el mundo laboral, promoviendo medidas para la reconversión y protección de los trabajadores ante los cambios tecnológicos y la automatización.

**6. Análisis de Requerimientos**

6.1 Síntesis de necesidades de cada perfil

Tenemos 6 roles de usuario para gestionar la biblioteca, empezando por el **Administrador del sistema**, quien configura, mantiene y mejora el sistema; seguido del **Bibliotecario**, encargado de préstamos, catálogo y sanciones; el **Usuario Estudiante**, que busca, reserva y consulta su historial de préstamos; el **Usuario Docente**, con acceso a material exclusivo, listas de lectura y plazos extendidos; el **Administrador de contenido**, responsable de documentos digitales, comentarios y eventos; y finalmente, el **Logístico**, quien supervisa el stock, adquisiciones, traslados e inventario de materiales.

6.2 Tabla de requisitos funcionales y no funcionales.



**7. Análisis del Sistema Actual**

7.1 Arquitectura general del sistema monolítico

En el sistema monolítico actual de la biblioteca, todas las funciones (gestión de libros, préstamos, usuarios, etc.) están integradas en una sola aplicación, lo que limita la flexibilidad, escalabilidad y mantenimiento del sistema. Las modificaciones y actualizaciones requieren cambios en toda la plataforma, lo que genera tiempos de inactividad y puede ocasionar fallos en otras áreas del sistema.

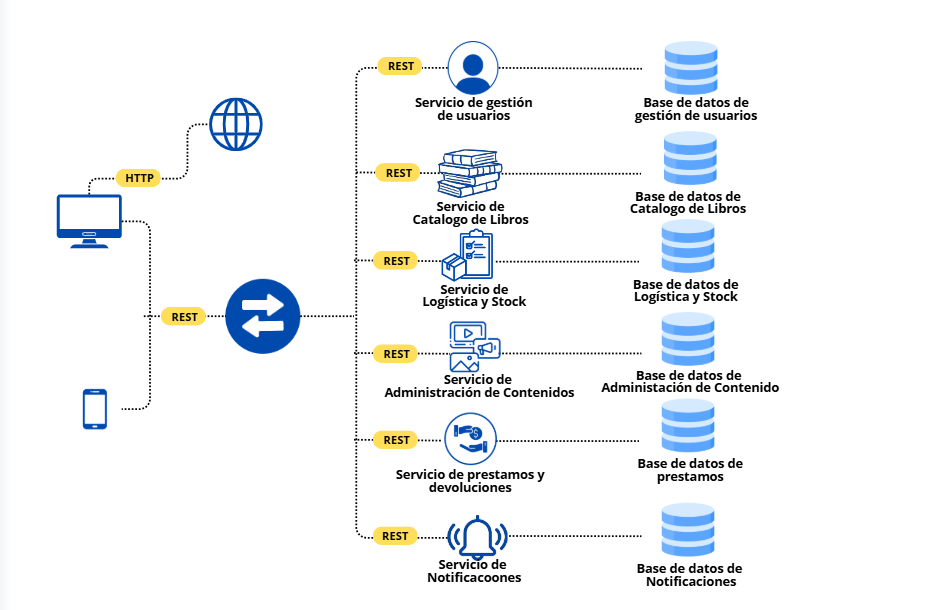
Este enfoque ha llevado a deficiencias en el rendimiento y la capacidad de adaptación a nuevas tecnologías, lo que ha impulsado la necesidad de migrar a una arquitectura basada en microservicios.

7.2 Puntos débiles detectados (fallas, cuellos de botella, mantenimiento)

* **Escalabilidad limitada** y afectación de todo el sistema ante un aumento de carga.
* **Mantenimiento costoso y riesgoso**, con impacto en otras partes del sistema.
* **Consultas lentas** debido a la base de datos centralizada.
* **Procesos congestionados** (préstamos y devoluciones) debido a la lentitud del sistema.
* **Falta de flexibilidad para implementar nuevas tecnologías** o servicios.
* **Código acoplado** que hace el mantenimiento y actualización difíciles y arriesgados.

Este diagnóstico pone de manifiesto la necesidad de migrar a una arquitectura más flexible y escalable, como la de microservicios, que permita superar estas limitaciones y mejorar la eficiencia del sistema.

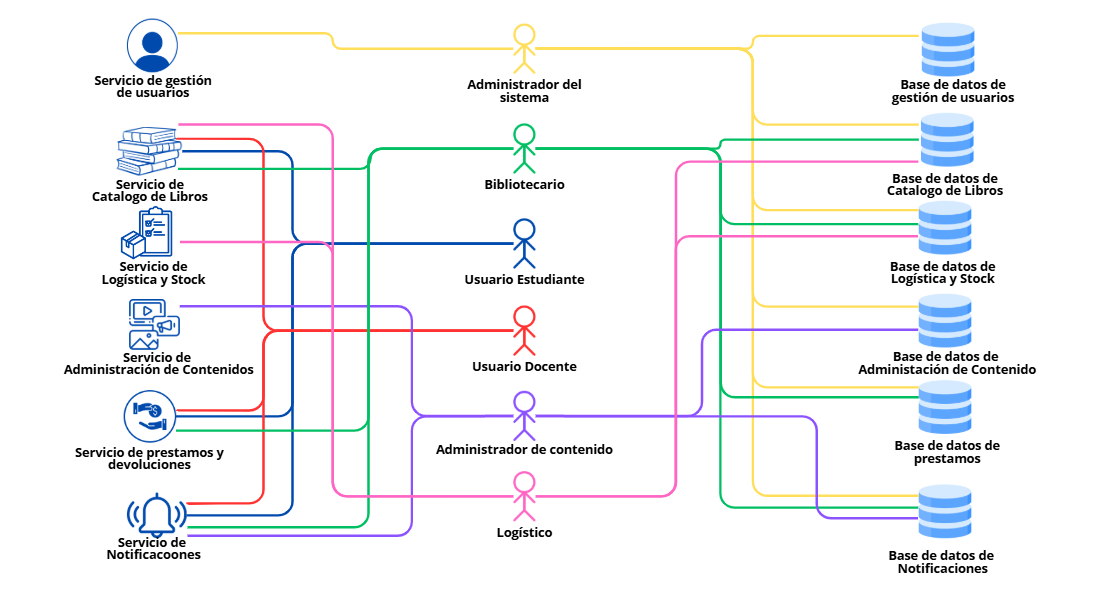
**8. Diseño de la Nueva Arquitectura**

8.1 Descripción general de la arquitectura basada en microservicios

8.2 Diagrama de Actores de Alto Nivel (quién interactúa con qué servicio) excel tabla



8.3 Diagrama de Casos de Uso



8.4 Diagrama de Clases



9.1 Fases de migración (por módulo o servicio)

### Fase 1: Análisis y Planificación (2-3 semanas)

* Evaluar el sistema monolítico actual y definir los servicios independientes.
* Identificar dependencias y definir la estrategia de integración entre microservicios.
* Diseñar la arquitectura general y elegir tecnologías (Docker, Kubernetes, API Gateway, etc.).
* Establecer un plan de despliegue y rollback para evitar interrupciones.

## Fase 2: Migración por Módulos

### 1. Microservicio de Autenticación y Gestión de Usuarios (3-4 semanas)

**Pasos:**

1. Extraer la lógica de autenticación y autorización del sistema monolítico.
2. Implementar un servicio de autenticación con JWT/OAuth2.
3. Configurar la base de datos de usuarios independiente.
4. Exponer endpoints para login, registro y roles.
5. Realizar pruebas de seguridad y carga.
6. Desplegar en producción con integración al sistema monolítico.

### 2. Microservicio de Catálogo de Libros (4-5 semanas)

**Pasos:**

1. Extraer la base de datos de libros y crear un servicio independiente.
2. Crear endpoints REST para CRUD de libros.
3. Configurar integración con autenticación y préstamos.
4. Probar rendimiento y tiempos de respuesta.
5. Desplegar en producción y redirigir las consultas del monolito a este servicio.

### 

### 

### 

### 3. Microservicio de Préstamos y Devoluciones (5-6 semanas)

**Pasos:**

1. Extraer la lógica de préstamos y transacciones.
2. Diseñar una base de datos distribuida para manejar la concurrencia.
3. Implementar lógica de disponibilidad de libros y validaciones.
4. Conectar con el servicio de usuarios y catálogo.
5. Configurar eventos para actualización en tiempo real.
6. Desplegar progresivamente con monitoreo.

### 4. Microservicio de Notificaciones (3-4 semanas)

### **Pasos:**

1. Implementar una cola de mensajes (RabbitMQ/Kafka) para manejar notificaciones asincrónicas.
2. Diseñar plantillas para notificaciones de préstamos, vencimientos y reservas.
3. Integrar con los servicios de préstamos y usuarios.
4. Habilitar notificaciones vía email y SMS.
5. Desplegar y probar con usuarios reales.

## Fase 3: Integración y Optimización (4-6 semanas)

1. Implementar API Gateway para gestionar la comunicación entre microservicios.
2. Realizar pruebas de estrés y seguridad en toda la arquitectura.
3. Optimizar la infraestructura con escalabilidad automática.
4. Capacitar a los equipos en la nueva arquitectura.

## 

## Fase 4: Desmantelamiento del Monolito (2-3 semanas)

1. Verificar que todos los microservicios funcionen de manera independiente.
2. Migrar datos finales y realizar pruebas de validación.
3. Desactivar gradualmente partes del sistema monolítico.
4. Monitorear el sistema en producción y solucionar incidencias.

### Tiempo Total Estimado: 6-8 meses

Este plan permite una transición progresiva y segura sin afectar la operatividad de la biblioteca.

9.2 Consideraciones de compatibilidad

Durante la migración del sistema monolítico a una arquitectura basada en microservicios, es fundamental garantizar la compatibilidad entre los nuevos servicios y los sistemas existentes. A continuación, se presentan las principales consideraciones para asegurar una transición sin interrupciones:

**1. Compatibilidad con el Sistema Monolítico Durante la Transición**

1. **Estrategia de migración gradual**: Durante la implementación de microservicios, el sistema monolítico seguirá operando parcialmente. Se deben crear interfaces que permitan la interoperabilidad entre ambos sistemas.
2. **Uso de API Gateway**: Un **API Gateway** permitirá redirigir ciertas solicitudes a los microservicios mientras otras continúan siendo manejadas por el monolito.
3. **Base de datos compartida (temporalmente)**: Durante la transición, algunos microservicios pueden acceder a la base de datos monolítica para evitar interrupciones en el servicio. Luego, cada microservicio tendrá su propia base de datos.

## 2. Compatibilidad con Bases de Datos

1. **Estrategia de migración de datos**:

* Se deben migrar los datos gradualmente sin afectar la disponibilidad del sistema.
* Uso de **ETL (Extract, Transform, Load)** para transformar datos del modelo monolítico al modelo distribuido.

1. **Implementación de sincronización temporal entre bases de datos.**

* **Consistencia eventual**: Se debe definir cómo manejar la consistencia de los datos entre múltiples microservicios sin afectar el rendimiento.
* **Compatibilidad con múltiples motores de bases de datos**: Cada microservicio puede usar una base de datos diferente (SQL o NoSQL), por lo que deben existir mecanismos de integración.

## 3. Compatibilidad con Servicios y Protocolos de Comunicación

1. **Soporte para REST y gRPC**: Los microservicios deben ofrecer APIs compatibles con REST para facilitar la integración con sistemas externos, pero también se puede considerar gRPC para una comunicación más eficiente entre microservicios.
2. **Mensajería asíncrona**: Uso de colas de mensajes (RabbitMQ, Kafka) para integrar microservicios sin depender de llamadas directas.
3. **Formatos de datos estándar**: JSON y XML deben ser los formatos estándar para la comunicación, asegurando la compatibilidad con clientes y sistemas externos.

## 4. Compatibilidad con Sistemas Externos

1. **Pasarelas de pago**: La nueva arquitectura debe poder integrarse con los sistemas actuales de pago de multas o cuotas.
2. **Interoperabilidad con bibliotecas externas**: Si la universidad consulta catálogos de otras bibliotecas, se debe garantizar la compatibilidad con los estándares de intercambio de datos (Z39.50, MARC, OpenAPI).

## 5. Compatibilidad con Dispositivos y Plataformas

1. **Accesibilidad web y móvil**: La nueva arquitectura debe garantizar que la plataforma web sea **responsive** y pueda utilizarse en dispositivos móviles.
2. **Compatibilidad con navegadores**: Asegurar soporte para navegadores modernos y versiones recientes de Chrome, Brave, Firefox, Edge y Safari.
3. **Optimización para rendimiento en dispositivos de bajo recurso**: Se debe minimizar el consumo de recursos para garantizar un buen desempeño en equipos con hardware limitado.

## 

## 6. Compatibilidad con Seguridad y Normativas

1. **Cumplimiento de normativas**: La nueva arquitectura debe cumplir con regulaciones de protección de datos como **GDPR, ISO 27001, y normativas de la universidad**.
2. **Manejo de credenciales y cifrado**: Todas las comunicaciones entre microservicios deben estar cifradas (TLS 1.2+).  
    **Compatibilidad con autenticación multifactor (MFA)**: Para mejorar la seguridad de los accesos.

Estas consideraciones aseguran que la migración sea compatible con los sistemas existentes, evitando interrupciones en el servicio y facilitando la integración de nuevas tecnologías.

9.2 Mecanismos de testing continuo explicar cómo estaremos monitoreando estos microservicios

#### 1. Pruebas Automatizadas (Continuous Integration/Continuous Testing - CI/CT)

* **Pruebas Unitarias**: Cada microservicio contará con un conjunto de pruebas unitarias que validen la correcta implementación de las funciones internas del servicio. Estas pruebas se ejecutarán automáticamente en el proceso de integración continua (CI) para asegurar que los cambios no introduzcan errores.
* **Pruebas de Integración**: Para verificar la comunicación y compatibilidad entre microservicios, se implementarán pruebas de integración automatizadas que validen que los servicios interactúan correctamente a través de sus APIs.
* **Pruebas de Regresión**: Se realizarán pruebas de regresión de manera continua para asegurarse de que las nuevas actualizaciones no rompan funcionalidades previamente implementadas, especialmente al migrar partes del sistema monolítico a los microservicios.
* **Pruebas de Carga**: Se simularán cargas de trabajo para evaluar cómo responden los microservicios bajo diferentes niveles de estrés y garantizar que puedan escalar adecuadamente según sea necesario.

#### 3. Monitoreo de Microservicios (Observabilidad)

La **observabilidad** es clave para identificar problemas rápidamente y asegurar la operatividad de los microservicios en producción. Los mecanismos de monitoreo incluirán:

* **Uso de Herramientas de Monitoreo**: Se implementarán herramientas como **Prometheus** y **Grafana** para la recolección de métricas en tiempo real sobre la salud de los microservicios. Estas herramientas permiten visualizar indicadores clave como tiempos de respuesta, uso de CPU, memoria, tasas de error y otros parámetros críticos.
* **Monitoreo de Logs**: Se utilizará una herramienta de centralización de logs como **ELK Stack** (Elasticsearch, Logstash, Kibana) o **Splunk** para recolectar, analizar y visualizar logs de todos los microservicios. Esto permite detectar errores, caídas o patrones inusuales en las interacciones entre servicios.
* **Alertas Automáticas**: Se establecerán alertas automáticas basadas en umbrales predefinidos (por ejemplo, alta latencia, error en respuestas, uso excesivo de recursos) que notificarán al equipo de desarrollo sobre posibles problemas antes de que afecten a los usuarios finales.
* **Tracing Distribuido**: Se implementará un sistema de tracing distribuido, como **Jaeger** o **Zipkin**, para rastrear solicitudes a través de los microservicios y detectar cuellos de botella, errores de comunicación o fallos en la integración entre servicios.
* **Análisis de Performance**: Herramientas como **New Relic** o **Datadog** se usarán para el análisis de rendimiento en tiempo real, permitiendo identificar rápidamente áreas que necesiten optimización.

#### 4. Testing de Seguridad Continuo

Para garantizar que los microservicios estén protegidos frente a vulnerabilidades y ataques, se integrarán pruebas de seguridad en el pipeline de CI/CD. Estas incluirán:

* **Escaneo de Vulnerabilidades**: Herramientas como **OWASP ZAP** o **Snyk** se utilizarán para realizar análisis de seguridad automatizados y detectar vulnerabilidades de seguridad en el código y dependencias.
* **Pruebas de Autenticación y Autorización**: Se validará que las políticas de seguridad, como el uso de JWT y OAuth2 para la autenticación, funcionen correctamente en todas las interacciones entre los microservicios.
* **Pruebas de Penetración**: De forma periódica, se realizarán pruebas de penetración para evaluar posibles vulnerabilidades en la infraestructura y las aplicaciones.

#### 5. Pruebas de Recuperación ante Desastres

Para garantizar la **resiliencia** del sistema, se realizarán pruebas de recuperación ante fallos (disaster recovery). Esto incluirá la simulación de fallos en microservicios y la validación de que el sistema se recupera adecuadamente sin afectar la operación de los demás servicios.

#### 6. Revisión y Optimización Continua

Tras el monitoreo y las pruebas, se revisarán de forma continua los resultados para identificar áreas de mejora en los microservicios. Esto incluye la optimización de bases de datos, la mejora de la eficiencia del código, y el ajuste de la infraestructura para adaptarse mejor a las cargas de trabajo reales.

9.3 Identificación de riesgos y su plan de mitigación explicar si ocurre un error como corregirlo

Durante la migración a microservicios, existen varios riesgos clave que pueden afectar el funcionamiento del sistema. A continuación, se describen los principales riesgos y las estrategias para mitigarlos:

1. **Incompatibilidad entre Microservicios y el Sistema Monolítico** **Riesgo**: Los microservicios deben integrarse gradualmente con el sistema monolítico.  
    **Mitigación**: Utilizar un **API Gateway** para gestionar la comunicación entre ambos y adoptar una migración gradual.  
    **Corrección**: Si ocurre un fallo, se debe revertir el despliegue y solucionar las interfaces antes de continuar.
2. **Problemas en la Comunicación entre Microservicios** **Riesgo**: Fallos de conexión o sincronización de datos entre microservicios.  
    **Mitigación**: Implementar pruebas de integración y mensajería asíncrona (RabbitMQ, Kafka).  
    **Corrección**: Si falla la comunicación, revisar la configuración de las APIs y las colas de mensajes.
3. **Pérdida de Datos durante la Migración** **Riesgo**: Pérdida o corrupción de datos al migrar de una base de datos monolítica a bases separadas por microservicios.  
    **Mitigación**: Usar procesos de migración gradual (ETL), realizar respaldos completos y sincronización temporal de bases de datos.  
    **Corrección**: Restaurar los datos desde un respaldo y corregir la causa del error.
4. **Falta de Escalabilidad de los Microservicios** **Riesgo**: Los microservicios no escalan adecuadamente bajo alta carga.  
    **Mitigación**: Realizar pruebas de carga y configurar escalabilidad automática para los microservicios.  
    **Corrección**: Si se detecta un problema, ajustar la infraestructura o mejorar el diseño del servicio.

En caso de error, se deben implementar procedimientos de reversión, monitoreo y solución rápida de problemas para asegurar que la transición se realice sin afectar la operatividad del sistema.

**10. Planificación Ágil y Cronograma**

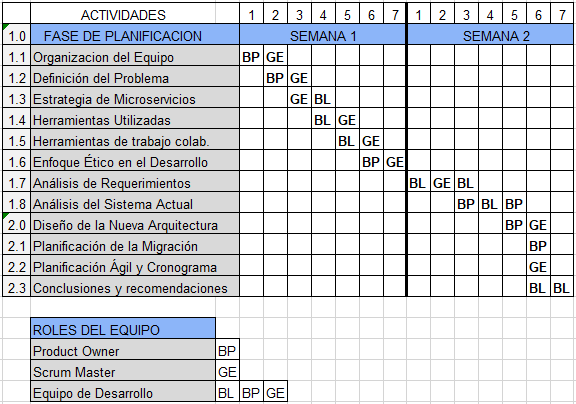
10.1 Breve explicación del enfoque ágil usado (Scrum, Kanban, etc.)

El proyecto utilizará el marco **Scrum** para desarrollar la migración a microservicios, lo que permite entregas rápidas, flexibilidad para adaptarse a cambios y una constante mejora del proceso. El cronograma se organiza en **sprints** de 2 semanas, con entregas incrementales que permiten implementar los microservicios de manera progresiva, asegurando que la transición del sistema monolítico a microservicios sea exitosa y eficiente.

Este enfoque ágil ayudará a garantizar que el sistema de gestión de la biblioteca evolucione de manera ordenada, manteniendo la calidad del servicio y minimizando los riesgos asociados con el cambio.

10.2 Roles asumidos por los integrantes roles de cada integrante

* **Product Owner (Propietario del Producto)**: Define las prioridades del proyecto y asegura que el equipo trabaje en las funcionalidades más valiosas y necesarias para la biblioteca. **Benjamín Palma**
* **Scrum Master**: Facilita el proceso Scrum, elimina obstáculos y asegura que el equipo siga las mejores prácticas ágiles. **Gustavo Espinoza**
* **Equipo de Desarrollo**: Grupo multidisciplinario encargado de implementar las funcionalidades acordadas, realizar pruebas y entregar los incrementos de trabajo al final de cada sprint. **Benjamín Leal**, **Gustavo Espinoza, Benjamín Palma**

10.3 Carta Gantt con cronograma de implementación y revisión.

**11. Conclusiones y recomendaciones**

11.1 Lecciones aprendidas

**Importancia de la Planificación Inicial**: Durante el proceso de migración del sistema monolítico hacia una arquitectura de microservicios, se aprendió que una **planificación exhaustiva** y una clara comprensión de los requisitos del sistema son fundamentales para evitar errores y retrasos durante el desarrollo. Definir correctamente los microservicios desde el principio permite que el equipo trabaje de manera organizada y eficiente.

**Gestión de la Complejidad**: La transición de un sistema monolítico a microservicios puede ser compleja, especialmente cuando se manejan bases de datos centralizadas y se requieren interacciones entre microservicios. Es crucial establecer una **estrategia clara de comunicación**

entre microservicios, para evitar dependencias cíclicas y problemas de escalabilidad.

**Comunicación y Colaboración en el Equipo**: A través del uso de **Scrum**, se destacó la importancia de la comunicación continua y la colaboración entre todos los miembros del equipo. Las reuniones diarias y la revisión continua del progreso aseguran que todos estén alineados con los objetivos del proyecto y permiten realizar ajustes oportunos.

**Adaptación a Nuevas Tecnologías**: La migración a microservicios permitió incorporar tecnologías emergentes, como bases de datos distribuidas y notificaciones automáticas. Aprendimos que las **tecnologías modernas** no solo mejoran el rendimiento, sino que también abren nuevas oportunidades para enriquecer los servicios ofrecidos a los usuarios.

11.2 Potencial escalabilidad del sistema

La **escalabilidad** es uno de los principales beneficios de adoptar una arquitectura de microservicios. Con la nueva arquitectura, el sistema de gestión de la biblioteca puede escalar de manera **independiente** cada componente según las necesidades de carga.

* **Escalabilidad Horizontal**: Los microservicios pueden ser replicados y distribuidos en diferentes servidores o instancias de manera eficiente, lo que mejora la capacidad de respuesta del sistema durante momentos de tráfico, como al inicio de cada semestre cuando la demanda de los estudiantes aumenta.
* **Escalabilidad de Funcionalidades**: A medida que se agregan nuevas funcionalidades, como análisis de datos o interfaces con otras plataformas, los microservicios permiten que estas se implementen sin afectar a las funcionalidades existentes. Esto asegura que la **capacidad de expansión** sea flexible y rápida.
* **Optimización de Recursos**: Con la separación de servicios, cada componente puede optimizarse de manera autónoma para cumplir con sus demandas específicas. Por ejemplo, el microservicio encargado de la gestión de libros puede requerir un sistema de base de datos más robusto, mientras que el de **notificaciones automáticas** puede ser más liviano en cuanto a infraestructura.

11.3 Posibles mejoras futuras

**Implementación de Inteligencia Artificial y Machine Learning**:

* **Recomendación**: Integrar **algoritmos de predicción** para predecir la demanda de libros, sugerir materiales de lectura según el historial de préstamos y realizar un análisis más profundo de las tendencias de los usuarios. Además, el uso de **Machine Learning** podría mejorar la gestión de préstamos al predecir las necesidades de reposición de libros.

**Mejora en la Interfaz de Usuario (UI/UX)**:

* **Recomendación**: Optimizar la **interfaz de usuario** para facilitar la navegación, especialmente en dispositivos móviles. Se pueden agregar funciones como un **chat en vivo** para consultas rápidas. La experiencia de usuario es clave, y con el paso del tiempo, las expectativas seguirán evolucionando.

**Integración con Servicios Externos**:

* **Recomendación**: Integrar el sistema de gestión de la biblioteca con plataformas externas, como **sistemas de pago en línea**, para gestionar multas por retrasos en las devoluciones o permitir reservas de libros con pago anticipado. También sería útil integrar herramientas de análisis de datos de uso para obtener informes más detallados.

**Automatización de Procesos Administrativos**:

* **Recomendación**: Implementar **sistemas automáticos de gestión de inventarios**, que notifiquen a los bibliotecarios cuando los libros estén cerca de agotarse o cuando se necesiten nuevas adquisiciones. Esto optimiza el flujo de trabajo y reduce la carga de tareas repetitivas.

**Mejoras en la Seguridad**:

* **Recomendación**: Integrar más funcionalidades de seguridad, como **autenticación multifactor** (MFA) para la protección de los datos de los usuarios, así como realizar auditorías de seguridad periódicas para detectar vulnerabilidades.

**Anexos**

• Tableros de planificación (Trello, Miro, etc.)

<https://trello.com/invite/b/67e1a05c274dbed596068d67/ATTI83aeca731e4ad8aaba88dcecb247e499F0ED1325/proyecto-semestral-biblioteca>

• Código fuente (enlace a repositorio)

<https://github.com/benjazJJ/Proyecto-Semestral-Biblioteca>