

# Digital Money House - Documento de Proyecto

*Elias Facundo Ledezma*

*email: [facled@yahoo.com.ar](mailto:facled@yahoo.com.ar)*

**Repositorio del proyecto:** <https://gitlab.com/jsFacled/dmh>

**Repo config server:** <https://github.com/jsFacled/dmh-configServer-repo>

**Workspaces en Postman:**

<https://web.postman.co/workspace/a9a27df8-e906-4216-8235-0f891f9a86d5>

## Objetivo final

El objetivo final del desafío es crear un repositorio de GitLab con el contenido programado, el link y el esbozo de la infraestructura utilizada. Además, es necesario **construir un documento**

de proyecto que contenga los siguientes ítems:

- Objetivos del proyecto.
- Planificación y descripción de las actividades (backlog) con plazos estimados.
- Informes de entrega.
- Informes de retro personal.
- Lecciones aprendidas.

---

## ■ Objetivos del proyecto

El proyecto Digital Money House tiene como objetivo el desarrollo de una plataforma de billetera virtual que permita a los usuarios realizar diversas operaciones financieras de manera segura y eficiente. Esta billetera virtual proporcionará funcionalidades clave como el registro de usuarios, la gestión de saldo, la transferencia de dinero y la gestión de medios de pago, todo dentro de una arquitectura basada en microservicios.

### **Objetivos principales:**

1. **Desarrollar un sistema de autenticación y autorización robusto:** Implementar servicios para registro, inicio de sesión y cierre de sesión utilizando JWT (JSON Web

- Token) y Spring Security, garantizando la seguridad en el acceso a la billetera.
2. **Crear una infraestructura de microservicios:** Diseñar y desarrollar microservicios independientes como ms-gateway, ms-authserver, ms-users, ms-accounts, entre otros, que trabajen de manera coordinada para ofrecer funcionalidades de la billetera. Cada microservicio estará enfocado en una funcionalidad específica, como la autenticación, la persistencia de datos de usuarios y la gestión de cuentas.
  3. **Ofrecer una cuenta virtual única:** Proveer a los usuarios de una Cuenta Virtual Uniforme (CVU) única para cada cuenta registrada. Esta cuenta estará asociada a un alias autogenerado y será el medio principal para gestionar sus operaciones financieras, incluidas las transferencias de dinero y la consulta de saldo.
  4. **Implementar la gestión de medios de pago:** Permitir a los usuarios registrar, consultar, actualizar y eliminar tarjetas de crédito o débito, con la finalidad de utilizarlas para cargar saldo en su billetera o realizar pagos.
  5. **Facilitar las transacciones monetarias:** Implementar servicios que permitan a los usuarios realizar depósitos de dinero desde tarjetas registradas y transferencias de saldo a otras cuentas o alias dentro o fuera del sistema, asegurando la integridad de las operaciones y el control de los fondos disponibles.
  6. **Registrar y visualizar la actividad financiera:** Proporcionar a los usuarios un historial completo de las transacciones (ingresos y egresos) realizadas a través de la billetera, con opciones para consultar los detalles de cada operación y aplicar filtros para una búsqueda más eficiente.

### Resultados esperados:

Al finalizar el proyecto, se espera que Digital Money House cuente con un conjunto de servicios API documentados con Swagger, que cubran las siguientes funcionalidades:

- Registro, inicio y cierre de sesión de usuarios.
- Gestión de perfil de usuario y tarjetas de crédito/débito.
- Ingreso de dinero a la billetera.
- Transferencias de dinero entre usuarios.
- Dashboard que muestra el saldo y los últimos movimientos.
- Historial completo de transacciones.

Este enfoque modular y basado en microservicios garantizará una arquitectura escalable y mantenible, apta para soportar el crecimiento futuro del producto.

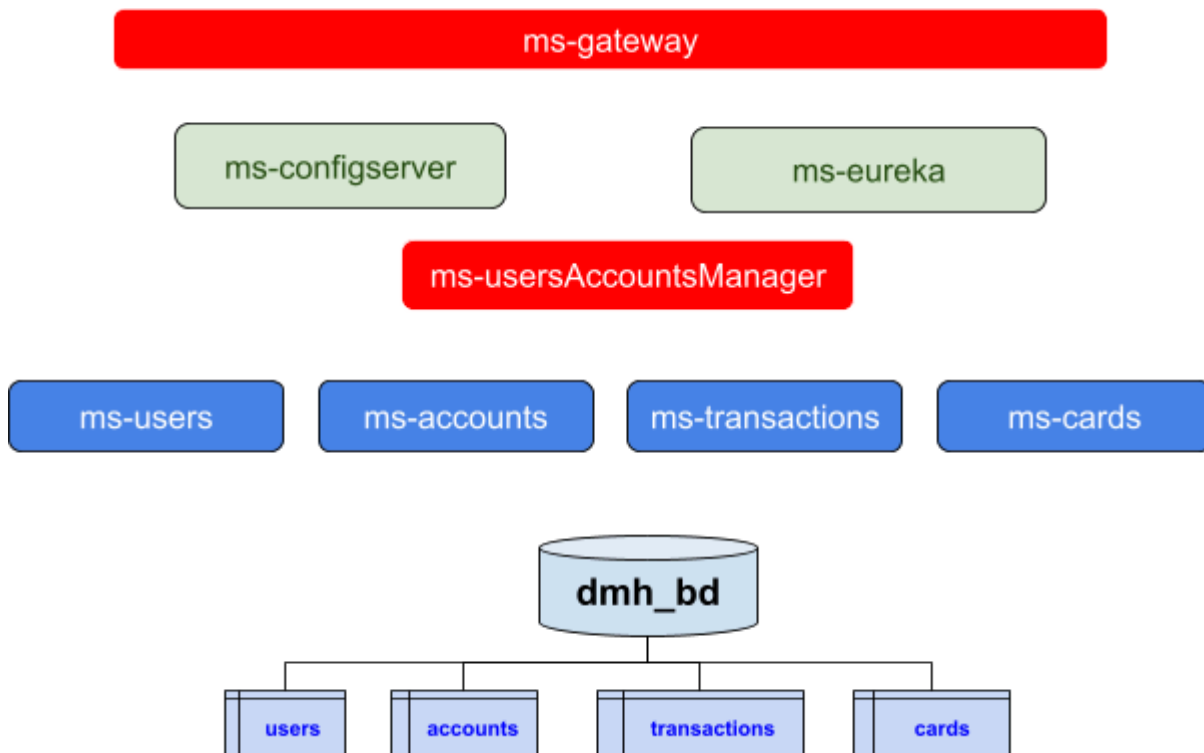
---

## ■ Planificación y descripción de las actividades (backlog) con plazos estimados.

Cada Spring dura 2 semanas.

El documento detallado se encuentra en el **anexo: “*Informes de entrega*”**.

# MICROSERVICIOS - ESTRUCTURA GENERAL



▶▶ **msvc-gateway:** Este microservicio actuará como el punto de entrada para todas las solicitudes del frontend. Realizará la autorización y autenticación al recibir JWT.

▶▶ **msvc-managerUserAccount:**

- Crear usuario y cuenta asociada.
- Login: Autenticación de usuarios y la generación de tokens JWT. Se comunicará con ms-users para validar las credenciales de los usuarios.

▶▶ **msvc-users:** Responsable de la persistencia y gestión de los datos de los usuarios. Proveerá servicios de registro, consulta y actualización de usuarios.

▶▶ **msvc-configserver:** Este microservicio gestionará la configuración centralizada de todos los microservicios.

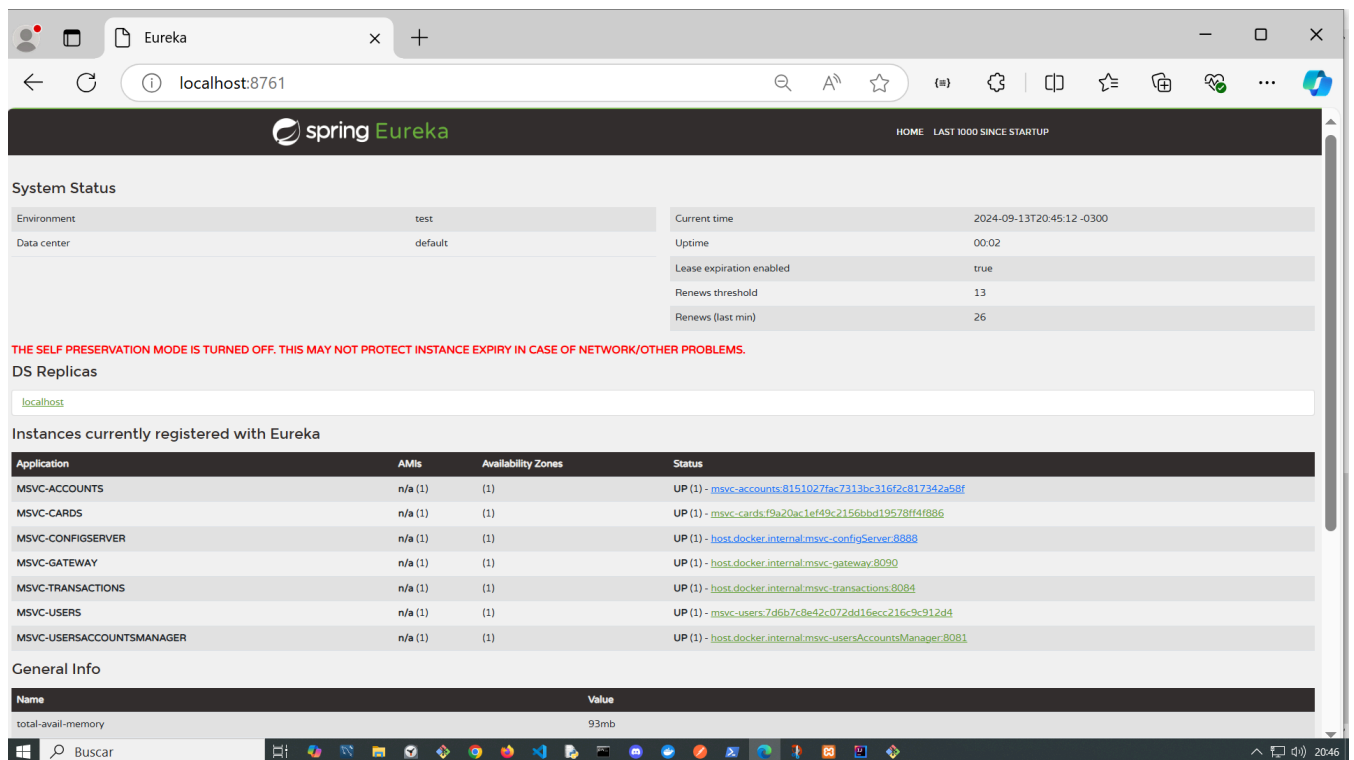
▶▶ **msvc-accounts:** Manejará todo lo relacionado con las cuentas de los usuarios, incluyendo el saldo, las transacciones y la actividad de la cuenta.

▶▶ **msvc-cards:** Gestionará las operaciones relacionadas con las tarjetas de crédito y débito asociadas a las cuentas de los usuarios.

▶ **msvc-transactions**: Encargado de registrar las transacciones que realiza msvc-accounts.

▶ **msvc-eureka**: registro de microservicios.

## Son 8 microservicios



The screenshot shows the Spring Eureka web interface. The top navigation bar includes the Spring Eureka logo and links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into several sections:

- System Status**: A table showing environment details (test, default) and system metrics (Current time, Uptime, Lease expiration enabled, Renew threshold, Renew last min).
- DS Replicas**: A section showing the local host as a replica.
- Instances currently registered with Eureka**: A table listing registered applications and their status.
- General Info**: A section showing system memory usage.

Application	AMIs	Availability Zones	Status
MSVC-ACCOUNTS	n/a (1)	(1)	UP (1) - <a href="#">msvc-accounts-8151027fac7313bc316f2c817342a58f</a>
MSVC-CARDS	n/a (1)	(1)	UP (1) - <a href="#">msvc-cards-f9a20ac1ef49c2156bbd19578f4f886</a>
MSVC-CONFIGSERVER	n/a (1)	(1)	UP (1) - <a href="#">host.docker.internal:msvc-configServer:8888</a>
MSVC-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">host.docker.internal:msvc-gateway:8090</a>
MSVC-TRANSACTIONS	n/a (1)	(1)	UP (1) - <a href="#">host.docker.internal:msvc-transactions:8084</a>
MSVC-USERS	n/a (1)	(1)	UP (1) - <a href="#">msvc-users-7d6b7c8e42c072dd16ecc216c9c912d4</a>
MSVC-USERSACCOUNTSMANAGER	n/a (1)	(1)	UP (1) - <a href="#">host.docker.internal:msvc-usersAccountsManager:8081</a>

Name	Value
total-avail-memory	93mb



## Ramas principales en gitlab



**Main:**

Está la documentación respaldatoria.

Correr cada microservicio en local desde consola gitbash con:

```
./mvnw spring-boot:run
```



**docker:**

Contiene **docker-compose** con configuración: `profiles: active: "dev"`



**dev:**

Correr cada microservicio en local desde consola gitbash con:

`./mvnw spring-boot:run`

## Test



**Postman:**



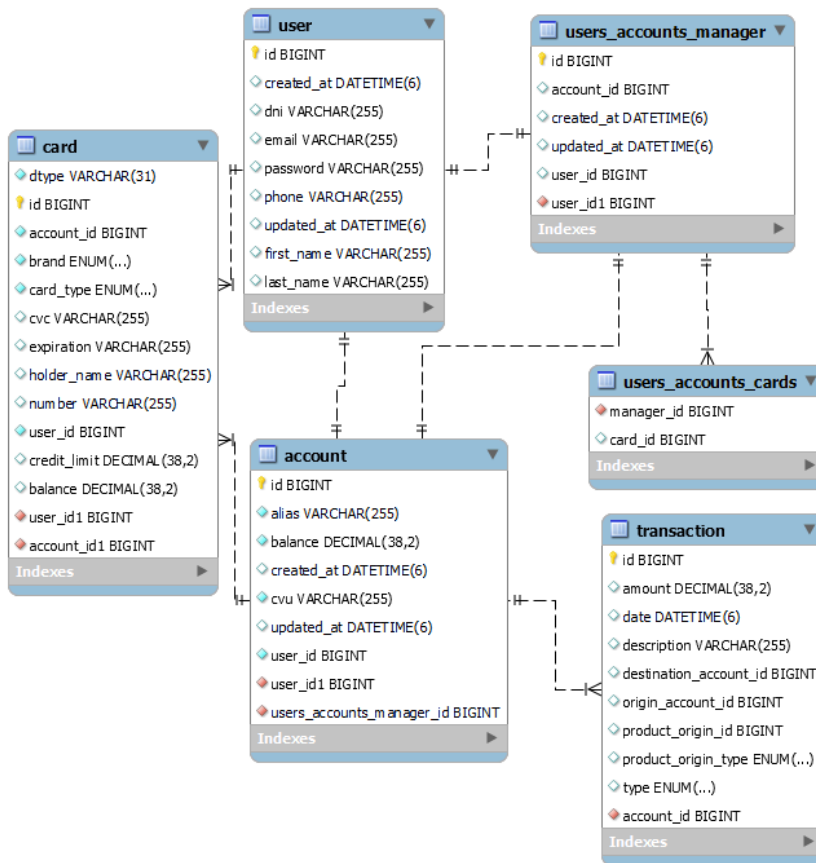
Se encuentra en el Anexo-Documentación: *“Informes de entrega”*.

## Infraestructura

Se encuentra en el Anexo-Documentación: *“Digital Money House-Estructura General”*.

## Base de Datos

Mysql dmh\_bd1



## ■ Informes de entrega.

Se encuentra en el Anexo-Documentación: *“Informes de entrega”*.

## ■ Informes de retro personal.

### Sprint 1:

#### 1. Lo que salió bien:

- **Cumplimiento de las funcionalidades de alta prioridad:** Se completaron todas las tareas relacionadas con el registro de usuario, inicio de sesión y cierre de sesión, que eran esenciales para la funcionalidad del negocio. Esto es un gran logro, ya que se garantizaron las funciones principales de la aplicación.
- **Diseño adecuado del modelo de datos:** El modelo de datos para el registro de

usuario fue diseñado e implementado con éxito, lo que facilitó el resto de las tareas relacionadas con la API de registro.

- **Manejo de errores eficaz:** Se implementaron correctamente los casos de error, lo que garantiza que la API pueda gestionar situaciones inesperadas como credenciales incorrectas o usuarios inexistentes.

## 2. Lo que podría mejorar:

- **Priorización de pruebas:** Aunque las funcionalidades principales están terminadas, las pruebas unitarias y de integración no se iniciaron durante este sprint. Esto es crítico para garantizar la calidad del software, por lo que en futuros sprints se debería dar más atención a las pruebas, incluso si su prioridad es más baja en esta etapa.
- **Estimación de tiempos:** Algunas tareas tomaron más tiempo de lo esperado, en particular aquellas relacionadas con el manejo de errores y la generación del CVU y alias ya que implicó la comunicación de varios microservicios.

## 3. Lo que aprendí:

- **Planificación efectiva:** Aprendí la importancia de dividir grandes tareas en subtareas manejables. Esto me ayudó a mantener un seguimiento claro del progreso y a asegurarme de que no se perdieran detalles importantes durante la implementación.
- **Flexibilidad con excepciones:** Durante el desarrollo, surgieron desafíos relacionados con el manejo de errores y las pruebas, lo que me mostró la importancia de ser flexible en la planificación para poder adaptarme a cambios inesperados.

## 4. Próximos pasos:

- **Priorizar las pruebas** unitarias y de integración en los siguientes sprints, ya que son esenciales para garantizar la estabilidad de las funcionalidades desarrolladas.
- **Ajustar el backlog** para incluir tareas relacionadas con la optimización de tiempos y mejorar la cobertura de pruebas.
- **Revisar las estimaciones de tiempo** de las tareas en el siguiente sprint para evitar retrasos y gestionar mejor las expectativas del equipo.

## 5. Conclusión: El sprint 1 fue exitoso en términos de entregar las funcionalidades clave de alta prioridad. Sin embargo, es fundamental que las pruebas y la cobertura de código se aborden en los próximos sprints para asegurar la estabilidad del sistema.

## Sprint 2:

### 1. Lo que salió bien:

- **Implementación de funcionalidades clave:** Se completaron con éxito las APIs relacionadas con el dashboard, perfil de usuario y la gestión de tarjetas, que eran de alta prioridad. Las funcionalidades del negocio avanzaron de acuerdo al plan.
- **Manejo correcto del saldo disponible y transacciones:** La API para obtener el saldo del usuario y sus últimos 5 movimientos fue implementada correctamente, asegurando que los usuarios puedan ver su resumen financiero de forma clara.
- **Avance en el CRUD de tarjetas:** Las funcionalidades de creación, eliminación y

consulta de tarjetas se desarrollaron sin mayores problemas, proporcionando al usuario la capacidad de gestionar sus tarjetas desde la plataforma.

## 2. Lo que podría mejorar:

- **Excepciones y errores en microservicios:** Aunque el CRUD de tarjetas está casi finalizado, el manejo de excepciones entre los microservicios aún requiere ajustes. Por ejemplo, la excepción 409 del microservicio **ms-cards** se está propagando incorrectamente como un error 500 en **ms-accounts**. Esto se debe manejar de forma más precisa para evitar confusiones en las respuestas al cliente.
- **Definir el esquema de respuesta para los movimientos:** Aún falta definir y completar el esquema de respuesta para los últimos 5 movimientos del usuario, lo cual es crucial para que el front-end muestre correctamente los datos al usuario.

## 3. Lo que aprendí:

- **Coordinación entre microservicios:** Este sprint mostró la importancia de una buena comunicación entre los microservicios. Cuando una API depende de otra para enviar respuestas precisas (como el manejo de errores entre **ms-cards** y **ms-accounts**), es esencial un control adecuado de excepciones.
- **Priorización de tareas críticas:** Aprendí que mantener el enfoque en las funcionalidades de alta prioridad relacionadas con el negocio permite avanzar sin descuidar los objetivos más importantes, aunque otras tareas de menor prioridad (como pruebas o manejo detallado de excepciones) queden para después.

## 4. Próximos pasos:

- **Optimizar el manejo de excepciones:** Abordar el problema con el manejo de errores entre **ms-cards** y **ms-accounts** para asegurar que el sistema devuelva el estado correcto en cada caso.
- **Definir el esquema pendiente para los movimientos:** Finalizar la definición del esquema de respuesta para los movimientos financieros de los usuarios, asegurando que el front-end reciba la información adecuada.
- **Revisar las pruebas:** Dado que las funcionalidades principales están terminadas, en el siguiente sprint debería centrarse en pruebas unitarias e integración, especialmente en los puntos críticos del CRUD de tarjetas y el dashboard.

## 5. Conclusión: El Sprint 2 fue exitoso en cuanto al desarrollo de las funcionalidades relacionadas con el dashboard y la gestión de tarjetas, cumpliendo con las prioridades del negocio. Sin embargo, es fundamental solucionar los problemas de manejo de errores entre microservicios y finalizar las partes pendientes para garantizar una experiencia fluida para los usuarios.

## Sprint 3:

### 1. Lo que salió bien:

- **Desarrollo de la funcionalidad de actividad:** Se completaron las APIs para ver todas las actividades y detalles de una actividad específica de los usuarios. La



lógica entre los microservicios **ms-accounts** y **ms-transactions** está bien implementada, proporcionando a los usuarios un historial detallado de sus transacciones.

- **Implementación del ingreso de dinero:** La funcionalidad clave para que los usuarios ingresen dinero desde sus tarjetas de crédito o débito fue desarrollada con éxito. Se implementaron correctamente las validaciones, la actualización de saldos y el registro de transacciones.
- **Manejo adecuado de errores comunes:** Las respuestas de estado **200 OK**, **400 Bad Request** y **404 Not Found** fueron implementadas en ambos microservicios (**ms-accounts** y **ms-transactions**), lo que mejora la robustez de la aplicación frente a entradas incorrectas o inexistentes.

## 2. Lo que podría mejorar:

- **Seguridad y autorización:** Aunque la lógica funcional está casi completa, el manejo de permisos y seguridad mediante Spring Security aún está en espera. Los estados **403 Forbidden** deben ser manejados para asegurar que solo los usuarios autorizados puedan acceder a su actividad o realizar transacciones.
- **Pruebas y manejo avanzado de excepciones:** El manejo de excepciones es correcto en los casos comunes, pero falta implementar una capa más avanzada para manejar excepciones críticas y excepcionales, especialmente en las transacciones entre microservicios (**ms-accounts** y **ms-transactions**).

## 3. Lo que aprendí:

- **Coordinación entre múltiples microservicios:** La implementación de funcionalidades que requieren comunicación entre varios microservicios es compleja pero clave para garantizar que las transacciones se registren y gestionen correctamente.
- **Flujo de transacciones:** Aprendí a optimizar el flujo de transacciones entre las tarjetas y las cuentas. Fue crucial ajustar los balances de las tarjetas (crédito y débito) y las cuentas, asegurando que la billetera funcione como se espera sin inconsistencias.

## 4. Próximos pasos:

- **Implementar seguridad con Spring Security:** El siguiente paso será activar la seguridad en las APIs mediante tokens y permisos adecuados, asegurando que los usuarios solo accedan a sus propios datos.
- **Manejo de errores avanzados y pruebas:** A medida que se completen las funcionalidades principales, se debe avanzar en mejorar las pruebas unitarias e integración, así como en el manejo de excepciones más detalladas.
- **Optimización y pruebas de carga:** Evaluar el rendimiento del sistema con grandes volúmenes de datos, especialmente en la visualización de la actividad completa del usuario.

## 5. Conclusión: El Sprint 3 avanzó de forma significativa, con las funcionalidades principales de ver actividades y el ingreso de dinero implementadas correctamente. La prioridad en el próximo sprint será fortalecer la seguridad y asegurar que las transacciones y actividades estén protegidas.

## Sprint 4:

### 1. Lo que salió bien:

- **Funcionalidad de transferencia de dinero:** Se completaron las APIs para consultar y realizar transferencias de dinero entre cuentas. Ahora los usuarios pueden transferir dinero desde su billetera a otra cuenta usando CBU/CVU o alias.
- **Consulta de destinatarios recientes:** La consulta para obtener los últimos 5 destinatarios de transferencias anteriores fue implementada con éxito. Se asegura que los destinatarios no se repitan y se devuelven los datos esenciales como ID, CVU y alias.
- **Actualización de saldos:** Se implementó correctamente la lógica para modificar los saldos de las cuentas de origen y destino. Además, se registra cada transacción de manera adecuada para mantener un historial completo.

## 2. Lo que podría mejorar:

- **Manejo de permisos:** El estado **403 Forbidden** sigue pendiente, lo que implica que aún no se ha integrado completamente la seguridad de Spring Security para validar los permisos en las transferencias de dinero.
- **Validaciones adicionales y manejo de errores:** El manejo de errores aún está incompleto para escenarios como cuentas inexistentes (**400 Bad Request**) y fondos insuficientes (**410 Gone**). Es necesario mejorar estas validaciones para asegurar una experiencia fluida para los usuarios.

## 3. Lo que aprendí:

- **Optimización de consultas:** La implementación de la consulta para los últimos destinatarios fue un desafío interesante, ya que implicaba asegurarse de que no se repitieran destinatarios y devolver solo los más recientes. Esto mejoró mi habilidad en la optimización de consultas en sistemas distribuidos.
- **Procesamiento de transacciones:** Aprendí a manejar el flujo de dinero entre cuentas de manera segura y eficiente, garantizando que las transacciones afecten correctamente los saldos de las cuentas involucradas.

## 4. Próximos pasos:

- **Seguridad y permisos en las transferencias:** Se debe integrar completamente la funcionalidad de seguridad y permisos para asegurar que solo usuarios autorizados puedan realizar transferencias.
- **Finalización del manejo de errores:** Completar las validaciones para los casos en los que la cuenta no existe o cuando no hay suficientes fondos. Estas son situaciones críticas que deben manejarse para prevenir errores en la aplicación.
- **Mejorar pruebas y excepciones:** Con la mayoría de las funcionalidades críticas ya implementadas, es un buen momento para concentrarse en mejorar las pruebas unitarias y de integración, así como el manejo avanzado de excepciones en las transacciones.

## 5. Conclusión: El Sprint 4 ha sido exitoso en la implementación de las funcionalidades principales de transferencias de dinero. Los siguientes pasos serán completar el manejo de permisos y validaciones para asegurar que la funcionalidad esté completamente operativa y segura antes del lanzamiento de la aplicación.

# ■ Lecciones aprendidas

## 1. **Planificación Efectiva**

Aprendí la importancia de dividir grandes tareas en subtareas manejables. Esta estrategia me permitió mantener un seguimiento claro del progreso y asegurarme de que no se perdieran detalles importantes durante la implementación. La planificación meticulosa y la segmentación de tareas son esenciales para la gestión eficiente de proyectos complejos.

## 2. **Flexibilidad con Excepciones**

Durante el desarrollo, enfrenté desafíos relacionados con el manejo de errores y las pruebas. Este proceso me enseñó la importancia de ser flexible en la planificación, permitiendo adaptaciones frente a cambios inesperados. La capacidad para ajustarse a las circunstancias y modificar el enfoque según sea necesario es crucial para el éxito del proyecto.

## 3. **Coordinación entre Microservicios**

He podido entender la importancia de una buena comunicación entre microservicios. Cuando una API depende de otra para enviar respuestas precisas (como el manejo de errores entre **ms-cards** y **ms-accounts**), es esencial un control adecuado de excepciones. La coordinación efectiva asegura que las interacciones entre los servicios sean precisas y fiables.

## 4. **Priorización de Tareas Críticas**

Aprendí que mantener el enfoque en las funcionalidades de alta prioridad relacionadas con el negocio permite avanzar sin descuidar los objetivos más importantes. Aunque otras tareas de menor prioridad (como pruebas o manejo detallado de excepciones) pueden quedar para después, es fundamental priorizar lo que impacta directamente en los objetivos del negocio.

## 5. **Coordinación entre Múltiples Microservicios**

La implementación de funcionalidades que requieren comunicación entre varios microservicios es compleja pero clave para garantizar que las transacciones se registren y gestionen correctamente. La integración efectiva de servicios interdependientes es crucial para el funcionamiento coherente del sistema.

## 6. **Flujo de Transacciones**

Aprendí a optimizar el flujo de transacciones entre tarjetas y cuentas. Ajustar los balances de las tarjetas (crédito y débito) y las cuentas fue crucial para asegurar que la billetera funcione sin inconsistencias. La gestión precisa de las transacciones es vital para mantener la integridad de los datos financieros.

## 7. **Optimización de Consultas**

La implementación de consultas para los últimos destinatarios presentó un desafío interesante. Asegurarse de que no se repitieran destinatarios y devolver solo los más recientes mejoró mi habilidad en la optimización de consultas en sistemas distribuidos. La eficiencia en la recuperación de datos es fundamental para la performance del sistema.

## 8. **Procesamiento de Transacciones**

Aprendí a manejar el flujo de dinero entre cuentas de manera segura y eficiente.

Garantizar que las transacciones afecten correctamente los saldos de las cuentas involucradas es esencial para mantener la integridad financiera del sistema.

#### 9. **Experiencia General en Microservicios**

La realización de un microservicio es solo una parte del desafío. Integrar un sistema completo implica enfrentar y resolver problemas relacionados con diversas tecnologías, como Docker, manejo de excepciones, bases de datos locales con MySQL y Hibernate, Spring Boot y sus proyectos asociados (Spring Cloud, Eureka Server, Spring Config, Gateway, Spring Security), el trabajo con JWTokens, documentación con Swagger, y despliegue en la nube con AWS. Además, el testing, las lecturas de logs y la gestión de errores son aspectos críticos que enriquecen la experiencia y el conocimiento en el desarrollo de sistemas distribuidos.