****

**Vives Bank**

**2º DAW IES LUIS VIVES**

**Jaime León Mulero**

**Germán Fernández Carracedo**

**Alba García Orduña**

**Natalia González Álvarez**

**Mario de Domingo Álvarez**

**Álvaro Herrero Tamayo**

**ÍNDICE**

1. **Introducción……………………………………………………………3**
2. **Requisitos……………………………………………………………….3**

**2.1 Requisitos No Funcionales…………………………..……………..3**

**2.2 Requisitos Funcionales………………………………..……..……..4**

**2.3 Requisitos de Información…………………………………………5**

1. **Diagramas……………………………………………………………….7**

**3.1 Diagrama de Clases………………………………………...………7**

1. **Explicación del Proyecto ……………………………………………8**
   1. **Tecnologías, enfoques y lenguajes usados………………….8**
   2. **Estructura………………………………………………………8**
   3. **Docker - compose……………………………………………10**
   4. **APIRest………………………………………………………12**
   5. **Notificaciones sucesos en tiempo real……………………….12**
   6. **Uso de Redis como caché en memoria……………………….14**
   7. **Endpoints………………………………………………………15**
   8. **Test……………………………………………………………..22**
2. **Organización:Trello…………………………………………………...23**
3. **Git Flow ……………………………………………………………….23**
4. **Costes …………………………………………………………………27**
5. **Bibliografía …………………………………………………………..29**

**Introducción**

En esta práctica vamos a realizar el servicio de un banco. Vamos a registrar los usuarios, los productos que contratan y sus movimientos.

Debemos tener en cuenta que un Cliente puede contratar una o varias cuentas y estas, a su vez, tienen una sola tarjeta. Por otro lado, contamos siempre con un administrador que puede acceder a la información del cliente y a sus movimientos.

Queremos realizar un programa para gestionar nuestro banco usando PostgreSQL para la base de datos relacional y MongoDB para la base de datos NoSQL.

En este proyecto nos basaremos en una arquitectura orientada al dominio, donde se aplican principios SOLID y se utilizan técnicas como Excepciones orientadas al dominio para garantizar la integridad y robustez del sistema.

**Requisitos**

**2.1 Requisitos No Funcionales:**

Son los requisitos que especifican los criterios que se utilizarán para evaluar el rendimiento del sistema, en lugar de las funciones específicas que debe realizar.

* RNF 1.- La exportación de los archivos de datos del cliente deberán ser de formato .json.
* RNF 2.- La exportación de los archivos de los movimientos del cliente deberán ser de formato .json y .pdf.
* RNF 3.- La copia de seguridad deberá comprimirse en un archivo de formato .zip.
* RNF 4.- Un cliente no deberá poder acceder al modo administrador.
* RNF 5.- Un administrador no deberá poder acceder al modo cliente.
* RNF 6.- No se podrá realizar ninguna operación de salida de dinero si no hay saldo suficiente en la cuenta.
* RNF 7.- No se podrá realizar ningún pago si supera alguno de los límites asignados.
* RNF 8.- El sistema de autenticación y autorización debe realizarse con JWT.
* RNF 9.- La gestión de los movimientos deberá realizarse usando MongoDB.
* RNF 10.- La documentación deberá realizarse con Swagger.
* RNF 11.- Deberán desplegarse todos los elementos del sistema usando Docker.
* RNF 12.- Se deberá usar excepciones adaptadas al dominio.
* RNF 13.- Todo el programa debe estar totalmente testeado con los casos correctos e incorrectos.
* RNF 14.- Realiza una arquitectura orientada al dominio.

**2.2 Requisitos Funcionales**

Son los requisitos que describen las funciones específicas que el sistema debe realizar. En el desarrollo de software, los requisitos funcionales suelen estar relacionados con las características y comportamientos del sistema.

RF 1.- Gestión de Usuarios

RF 1.1.- Crear usuario

RF 1.2.- Buscar usuario

RF 1.3.- Actualizar usuario

RF 1.4.- Borrar usuario

RF 1.5.- Gestión de Clientes

RF 1.5.1.- Crear cliente

RF 1.5.2.- Buscar cliente

RF 1.5.3.- Modificar cliente

RF 1.5.4.- Eliminar cliente

RF 1.6.- Gestión de Administradores

RF 1.6.1.- Crear administrador

RF 1.6.2.- Buscar administrador

RF 1.6.3.- Modificar administrador

RF 1.6.4.- Eliminar administrador

RF 2.- Gestión de Productos

RF 2.1.- Gestión de Cuentas

RF 2.1.1.- Crear cuenta

RF 2.1.2.- Buscar cuenta

RF 2.1.3.- Mostrar cuentas

RF 2.1.4.- Modificar cuenta

RF 2.1.5.- Eliminar cuenta

RF 2.2.- Gestión de Tarjetas

RF 2.2.1.- Crear tarjeta

RF 2.2.2.- Buscar tarjeta

RF 2.2.3.- Mostrar tarjetas

RF 2.2.4.- Modificar tarjeta

RF 2.2.5.- Eliminar tarjeta

RF 3.- Registro de Movimientos

RF 4.- Gestión Operaciones

RF 4.1.- Domiciliación de recibos

RF 4.2.- Ingresos de nóminas

RF 4.3.- Ingresos

RF 4.4.- Transferencias

RF 4.4.1- Revocar transferencias

RF 4.5.- Pagos de tarjetas

RF 5.- Realizar copia de seguridad

RF 6.- Obtener datos del cliente

RF 7.- Obtener movimientos del cliente

RF 8.- Obtener datos del cliente y sus productos

RF 9.- Realizar cambio de divisas

RF 10.- Sistema de notificaciones a tiempo real

RF 11.- Consultar listado de cajero

**2.3 Requisitos de Información:**

Son los requisitos que especifican los datos que el sistema debe almacenar, procesar o manipular.

RI 1.- El Cliente tendrá los siguientes campos:

- Id

- Nombre Completo

- Dirección

- Email

- Teléfono

- DNI

- Username

- Contraseña

- Foto

- Foto DNI

- Created At

- Updated At

- Is Delete

RI 2.- El Tipo de Cuenta tendrá los siguientes campos:

- Nombre

- Interés

RI 3.- La Cuenta tendrá los siguientes campos:

- Id

- IBAN

- Saldo

- Fecha Apertura

- Nombre

- Interés

- Created At

- Updated At

- Is Delete

RI 4.- El Tipo de Tarjeta tendrá los siguientes campos:

- Tipo

RI 5.- La Tarjeta tendrá los siguientes campos:

- Id

- Número

- Fecha Caducidad

- CVV

- PIN

- Gasto Diario

- Gasto Semanal

- Gasto Mensual

- Tipo

- Created At

- Updated At

- Is Delete

RI 6.- Los Movimientos tendrán los siguientes campos:

- Id Cliente

- Operación

RI 7.- La Operación tendrá los siguientes campos:

- Fecha

- Importe

RI 8.- Las Operaciones de Entrada tendrán los siguientes campos:

- Destino

RI 9.- Las Operaciones de Salida tendrán los siguientes campos:

- Destino

RI 10.- Las Operaciones de Entrada y Salida tendrán los siguientes campos:

- Destino

- Origen

RI 11.- El Administrador tendrá los siguientes campos:

- Id

- Username

- Contraseña

- Created At

- Updated At

- Is Deleted

RI 12.- Los movimientos estarán compuestos de operaciones.

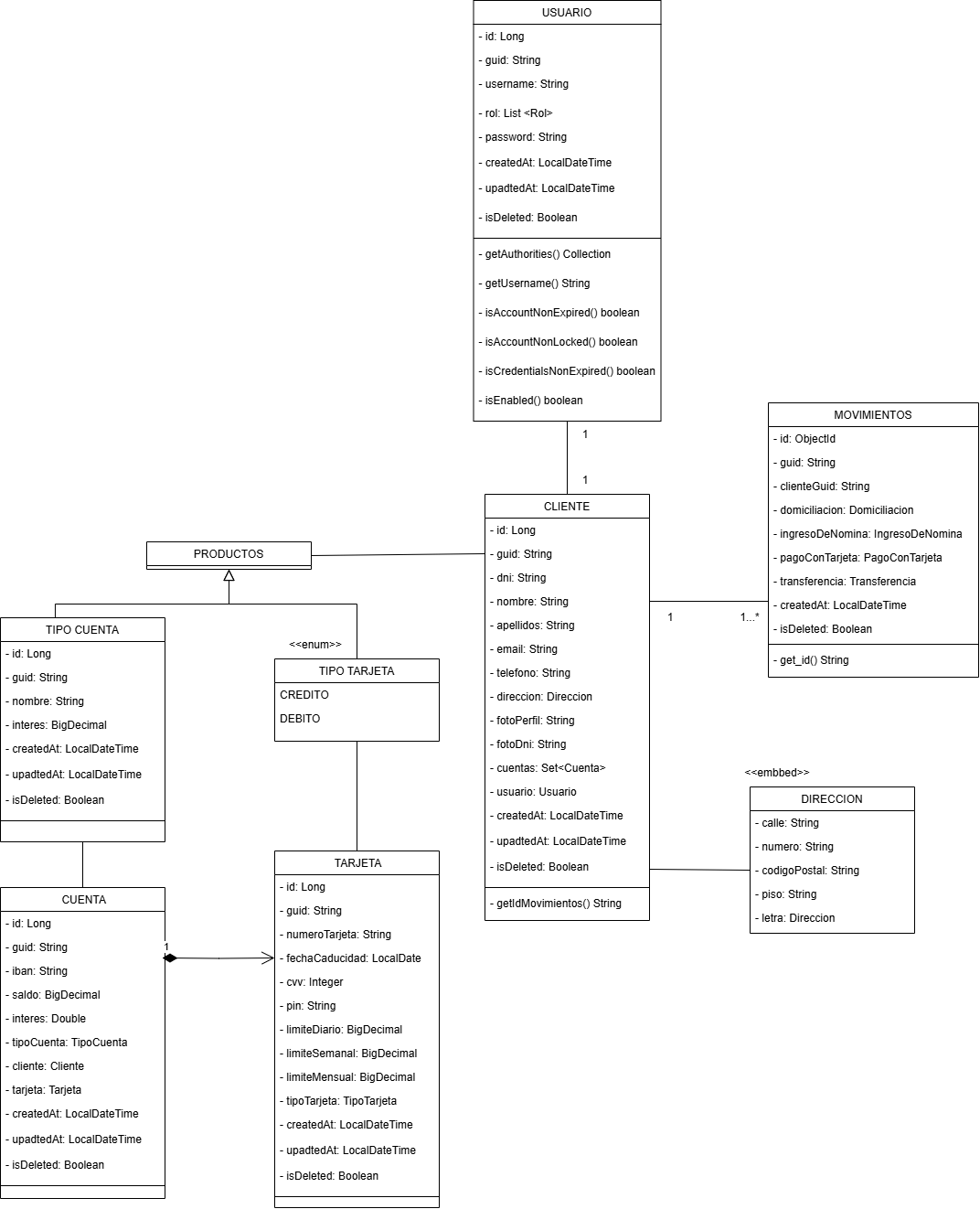
RI 13.- El cliente puede contratar una cuenta que puede ser normal, cuyo interés es del 2% o de ahorro cuyo interés es del 1%.

RI 14.- Una cuenta está compuesta por una tarjeta, que puede ser de crédito o de débito.

RI 15.- El administrador puede modificar el interés de las cuentas.

**Diagrama de Clases**

Un diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos) y las relaciones entre los objetos.

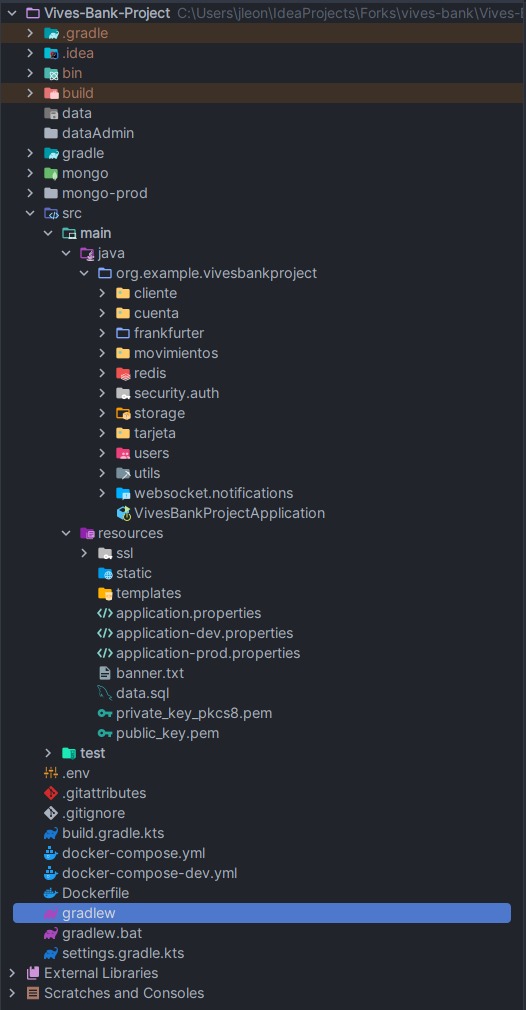
****

**Funcionamiento del Programa**

**Tecnologías, enfoques y lenguajes usados:**

* Arquitectura orientada al dominio.
* Manejo de excepciones orientadas al dominio.
* Uso de inyección de dependencias.
* Java
* Docker
* PostgreSQL
* MongoDB
* Postman
* Retrofit
* Git
* GitFlow
* Gradle
* SLF4J con Logback
* Swagger
* JUnit
* Mockito
* TestContainers

**Estructura:**

****

**Carpeta java**

En esta carpeta está el código fuente del proyecto, organizado en paquetes que siguen la estructura de un dominio.

* *cliente*: Contiene controladores, servicios y repositorios que gestionan los datos del cliente.
* *cuenta*: Aquí se manejan las cuentas bancarias de los clientes, como saldos o información de las cuentas.
* *frankfurter*: Este paquete está relacionado con la API de Frankfurter, que se usa en el proyecto para el intercambio de divisas. Aquí están las clases que gestionan la comunicación con esa API.
* *movimientos*: Contiene clases que registran o gestionan los movimientos de las cuentas bancarias. Como las transferencias, los pagos con tarjeta, los ingresos de nóminas o las domiciliaciones.
* *redis*: Este paquete está vinculado al uso de Redis para manejar cachés en el proyecto.
* *security.auth*: Está relacionado con la autenticación y seguridad del sistema, como el manejo de tokens o credenciales de usuario.
* *storage*: Es donde se gestionan archivos o datos persistentes relacionados con el banco. Aquí se generan los archivos JSON o PDF, se leen los ficheros CSV y se guarda la copia de seguridad en formato ZIP.

Además contamos con las carpetas *data* y *dataAdmin* donde se guardan todos los ficheros respectivos de cada usuario.

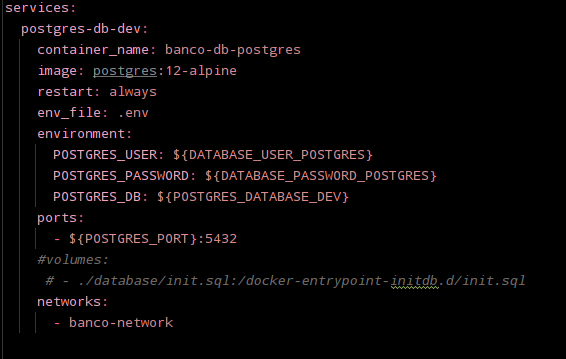
* *tarjeta*: Gestiona las tarjetas de los clientes, como crédito o débito, y sus operaciones.
* *users*: Aquí se manejan los usuarios del sistema, administradores o clientes que interactúan con el sistema.
* *utils*: Contiene utilidades generales que pueden ser usadas en todo el proyecto, como funciones de generadores o herramientas de ayuda.
* *websocket.notifications*: Está relacionado con notificaciones en tiempo real usando WebSockets, por ejemplo, para avisar a los usuarios de nuevas transaccione.
* *VivesBankProjectApplication*: Es la clase principal de la aplicación, desde donde se inicia el proyecto (normalmente contiene el método main).

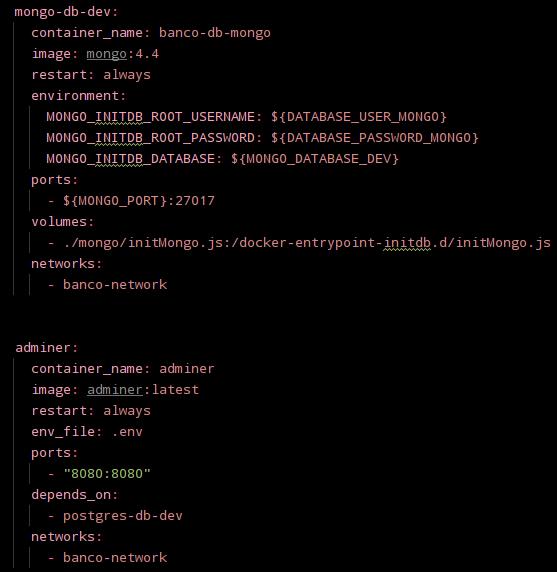
### **Carpeta resources**

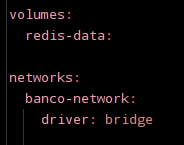
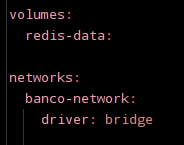
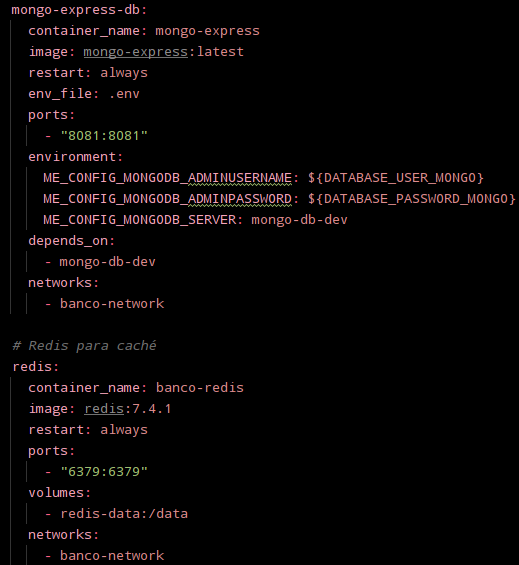
Aquí se almacenan los recursos estáticos y configuraciones del proyecto.

* *ssl*: Contiene archivos relacionados con certificados de seguridad, como claves privadas y públicas necesarias para comunicaciones seguras.
* *application.properties y variantes (-dev, -prod)*: Archivos de configuración donde se definen propiedades importantes del proyecto, como conexiones a bases de datos, puertos, o configuraciones de perfiles (desarrollo y producción).
* *banner.txt*: Contiene un mensaje personalizado que aparece al iniciar la aplicación.
* *data.sql*: Un archivo SQL que se usa para inicializar datos en la base de datos, como datos de prueba o tablas necesarias.
* *Claves (private\_key\_pkcs8.pem, public\_key.pem)*: Archivos de claves que se usan para firmar o verificar datos, garantizando la seguridad de las operaciones.

**Docker-Compose:**

Un archivo Docker Compose es una herramienta que permite definir y ejecutar aplicaciones que constan de múltiples contenedores Docker. Utiliza un archivo de configuración en formato YAML para especificar los servicios, redes y volúmenes necesarios para que la aplicación funcione correctamente.





**Configuración de la base de datos PostgreSQL**

Uno de los servicios principales es postgres-db-dev, que configura una base de datos PostgreSQL. Esta base de datos utiliza la imagen postgres:12-alpine, una versión optimizada para reducir el consumo de recursos. Gracias al uso de un archivo .env, podemos definir fácilmente el usuario, la contraseña y el nombre de la base de datos sin incluir información sensible directamente en el archivo de configuración.

El servicio está configurado para exponer el puerto interno de PostgreSQL (5432) al puerto definido en la variable de entorno ${POSTGRES\_PORT}. Esto nos permite acceder a la base de datos desde nuestro equipo local o desde otros servicios dentro del proyecto.

**Configuración de la base de datos MongoDB**

Para manejar datos no relacionales, el servicio mongo-db-dev implementa una base de datos MongoDB basada en la imagen mongo:4.4. Este contenedor también utiliza variables de entorno para establecer el usuario, la contraseña y la base de datos inicial. Además, incluye un volumen que mapea un archivo local llamado initMongo.js. Este archivo permite inicializar MongoDB con configuraciones o datos necesarios al momento de arrancar.

MongoDB expone su puerto estándar (27017) al puerto especificado en ${MONGO\_PORT}, lo que facilita el acceso a la base de datos desde herramientas externas o servicios relacionados.

**Implementación de Redis como sistema de caché**

El sistema de caché del proyecto está basado en Redis, un almacén en memoria de alto rendimiento. El servicio utiliza la imagen redis:7.4.1 y expone su puerto estándar (6379). Para garantizar la persistencia de los datos, Redis utiliza un volumen llamado redis-data. Esto asegura que, incluso si el contenedor se detiene o reinicia, los datos almacenados no se perderán.

#### **Conexión entre los servicios**

Todos los servicios están conectados mediante una red personalizada llamada banco-network. Esto permite que los contenedores se comuniquen entre sí sin necesidad de exponer puertos adicionales al exterior, aumentando la seguridad y el aislamiento del sistema. Además, la configuración asegura que los servicios dependientes, como Adminer o Mongo Express, solo se inicien cuando las bases de datos que necesitan estén listas.

**ApiRest:**

API REST (Application Programming Interface - Representational State Transfer) es un modelo que permite que diferentes aplicaciones se comuniquen a través de internet enviando datos en formatos estándar como JSON. Es ampliamente usada porque facilita la integración entre sistemas, permitiendo que servicios y aplicaciones trabajen juntos de manera eficiente.

En nuestro proyecto, hemos utilizado la API Frankfurter (<https://frankfurter.dev/>) para gestionar el intercambio de divisas. Esta API nos permite obtener tasas de cambio actualizadas y realizar conversiones de moneda de forma precisa. Por ejemplo, enviamos solicitudes indicando la moneda de origen, la moneda destino y el monto a convertir, y la API nos devuelve los resultados instantáneamente.

**Comunicaciones en tiempo real de sucesos del sistema. Notificaciones con Websockets.**

Las comunicaciones en tiempo real de sucesos del sistema se han implementado mediante WebSockets.

Un WebSocket es un protocolo de comunicación bidireccional en tiempo real utilizado en servicios web para habilitar la comunicación entre cliente y servidor mediante una conexión persistente.

Los eventos que vamos a notificar son los relacionados con movimientos bancarios: Domiciliación, Ingreso en Nómina, Pago con Tarjeta, Transferencia (y revocación). Igualmente notificaremos cambios en Cuentas y en Tarjetas de Crédito.

Por ejemplo, si se realiza una transferencia, se notificará tanto la creación de ese movimiento de transferencia, como la actualización del saldo de las cuentas implicadas.

Y si se efectúa el pago de una operación domiciliada en la fecha prevista, se notificará la ejecución de dicha operación, además de la creación de los movimientos y actualizaciones de la cuenta afectada.

Por tanto, se han tipificado las notificaciones en función de la operación realizada. Los tipos son: CREATE, UPDATE, DELETE, EXECUTE.

**Notificación a usuario específico**

En nuestra aplicación, notificaremos únicamente a los usuarios afectados por el movimiento bancario en cuestión. Por ejemplo, notificaremos únicamente al cliente origen y al cliente destino de una transferencia, independientemente de que haya otros clientes logueados con su usuario respectivo.

La notificación se hará además, comprobando si el usuario destino de la notificación se encuentra logueado en ese momento.

**Cómo se ha implementado**

En la clase **WebSocketHandler**, utilizamos una estructura (en este caso un mapa) para asociar cada sesión de WebSocket con el nombre de usuario autenticado y así enviar mensajes a las sesiones correctas.

Al establecer la conexión (**afterConnectionEstablished**), obtenemos el nombre del usuario autenticado y asociamos la sesión de WebSocket con el nombre de usuario en el mapa de sesiones-usuarios.

Al cerrar la conexión (**afterConnectionClosed**), eliminamos la sesión de la lista de sesiones y del mapa sesiones-usuarios.

El envío de mensajes a usuarios específicos se hará mediante el método **sendMessageToUser** que envía mensajes a un usuario concreto, cuyo nombre de usuario le pasaremos por parámetro.

Este recuperará del mapa de sesiones-usuarios la sesión del usuario específico y, siempre que esté abierta ( es decir, que el usuario esté conectado a las notificaciones), se le enviará la notificación.

**Contexto de seguridad y WebSockets. Uso de HandshakeInterceptor**

Spring Security no comparte contexto con WebSocket. Por esta razón, si las sesiones de WebSocket no heredan el contexto de seguridad correctamente, el SecurityContextHolder puede no contener la autenticación adecuada. Esto ocurre porque Spring Security no gestiona nativamente la autenticación en conexiones WebSocket.

Como solución se utilizará un HandshakeInterceptor en cada WebSocketHandler para propagar el contexto de seguridad a la sesión de WebSocket.

Un interceptor de seguridad (**HandshakeInterceptor**) es un componente de Spring, que comprueba si el usuario está autenticado antes de aceptar la conexión WebSocket. Si lo está, almacena el nombre de usuario en los atributos de la sesión WebSocket para futuras referencias. Si no lo está, la conexión es rechazada.

Así, al establecer la conexión en el WebSocketHandler, podremos acceder correctamente al nombre de usuario almacenado en los atributos de la sesión.

Cuando registramos nuestros WebSocketHandlers, lo hacemos añadiendo a cada uno un interceptor de seguridad.

**Funcionamiento de las notificaciones.**

Para emitir la notificación, en el servicio correspondiente a la entidad, hemos implementado un método onChange, que llamamos cada vez que se produce un cambio que debamos notificar.

Por ejemplo, en el servicio de Cuentas lo llamaremos desde los métodos save, update, delete, con el tipo de notificación correspondiente, por ejemplo, CREATE para el método save.

El método onChange, lo que hace básicamente es componer una cadena con la notificación a enviar y averiguar el nombre de usuario del cliente afectado por el cambio (en este caso el dueño de la cuenta). Con ello, llama al método de envío a usuario específico de nuestro servicio de WebSocket.

Este método es el que como hemos explicado antes, recupera del mapa de sesiones-usuarios la sesión del usuario específico y le enviará la notificación si está conectado.

**Uso de redis como caché en memoria**

El sistema de caché del proyecto está basado en Redis, un almacén en memoria de alto rendimiento. Para utilizarlo, se configura la conexión con el servidor de Redis y un cliente de Redis.

Los valores de conexión al servidor se almacenan en el fichero **application.properties** para el hostname y el puerto, y se recuperan al crear el bean de configuración del servidor:

spring.data.redis.host=${REDIS\_HOST:localhost}

spring.data.redis.port=${REDIS\_PORT:6379}.

Para crear y configurar el cliente para operaciones de acceso a datos con Redis, se crea un bean RedisTemplate y se parametriza. En nuestro caso el cliente está configurado para serializar objetos JSON y para habilitar el soporte para transacciones.

**Uso**

Para utilizar la caché debemos habilitarla en la clase de inicio de nuestra aplicación, mediante la anotación @EnableCaching, y después, en cada método que deba ser cacheado, utilizaremos las anotaciones:

@Cacheable (que salta el método de búsqueda si ya se buscó previamente con los mismos argumentos).

@CachePut (no salta el método sino que lo invoca y se guarda el resultado en la caché).

@CacheEvict (el método anotado desencadena una eliminación en la caché).

**Endpoints:**

Clientes

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todos los clientes | /clientes | GET | Requiere autenticación | Obtiene todos los clientes | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener cliente por ID | /clientes/{id} | GET | Requiere autenticación | Obtiene un cliente por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Catálogo | /clientes/catalogo | GET | Requiere autenticación | Obtiene todos los productos que puede contratar el cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear cliente | /clientes | POST | Requiere autenticación | Crea un nuevo cliente | 201 Created | 400 Bad Request, 409 Conflict |
| Actualizar cliente | /clientes/{id} | PUT | Requiere autenticación | Actualiza un cliente por su ID | 200 OK | 400 Bad Request, 404 Not Found |
| Eliminar cliente | /clientes/{id} | DELETE | Requiere autenticación | Elimina un cliente por su ID | 204 No Content | 404 Not Found |
| MeProfile | /clientes/me/perfil | GET | Requiere autenticación | Obtiene el perfil del cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| MeUpdate | /clientes/me/perfil | PUT | Requiere autenticación | Actualiza el perfil del cliente | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden,404 Not Found |
| Derecho Al Olvido | /clientes/me/perfil | DELETE | Requiere autenticación | Elimina el perfil del cliente | 200 OK | 401 Unauthorized, 403 Forbidden,404 Not Found |

Cuentas

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todas las cuentas | /cuentas | GET | Requiere autenticación | Obtiene todas las cuentas | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener cuenta por ID | /cuentas/{id} | GET | Requiere autenticación | Obtiene una cuenta por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener todas por ID del cliente | /cuentas/cliente/{id} | GET | Requiere autenticación | Obtiene una cuenta por el ID del cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear cuenta | /cuentas | POST | Requiere autenticación | Crea una nueva cuenta | 201 Created | 400 Bad Request, 409 Conflict |
| Actualizar cuenta | /cuentas/{id} | PUT | Requiere autenticación | Actualiza una cuenta por su ID | 200 OK | 400 Bad Request, 404 Not Found |
| Eliminar cuenta | /cuentas/{id} | DELETE | Requiere autenticación | Elimina una cuenta por su ID | 204 No Content | 404 Not Found |

Tipo de Cuentas

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todos los tipos | /tipocuentas | GET | Requiere autenticación | Obtiene todos los tipos de cuentas | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener tipo por ID | /tipocuentas/{id} | GET | Requiere autenticación | Obtiene un tipo de cuenta por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear tipo de cuenta | /tipocuentas | POST | Requiere autenticación | Crea un nuevo tipo de cuenta | 201 Created | 400 Bad Request, 409 Conflict |
| Actualizar tipo de cuenta | /tipocuentas/{id} | PUT | Requiere autenticación | Actualiza un tipo de cuenta por su ID | 200 OK | 400 Bad Request, 404 Not Found |
| Eliminar tipo de cuenta | /tipocuentas/{id} | DELETE | Requiere autenticación | Elimina un tipo de cuenta por su ID | 204 No Content | 404 Not Found |

Movimientos

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todos los movimientos | /movimientos | GET | Requiere autenticación | Obtiene todos los movimientos | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener un movimiento por su ID | /movimientos/id/{id} | GET | Requiere autenticación | Obtiene un movimiento por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener movimientos por el GUID del cliente | /movimientos/cliente/{guid} | GET | Requiere autenticación | Obtiene movimientos por el GUID del cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear movimiento | /movimientos | POST | Requiere autenticación | Crea un movimiento | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden |
| Crear ingreso nómina | /me/ingresonomina | POST | Requiere autenticación | Crea un ingreso de nómina | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden |
| Crear pago tarjeta | /me/pagotarjeta | POST | Requiere autenticación | Crea un pago de tarjeta | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden |
| Crear transferencia | /me/transferencia | POST | Requiere autenticación | Crea una transferencia | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden |
| Crear domiciliación | /me/domiciliacion | POST | Requiere autenticación | Crea una domiciliación | 200 OK | 400 Bad Request, 401 Unauthorized, 403 Forbidden |

Tarjetas

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todas las tarjetas | /tarjetas | GET | Requiere autenticación | Obtiene todas las tarjetas | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener tarjeta por ID | /tarjetas/{id} | GET | Requiere autenticación | Obtiene una tarjeta por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener datos privados de tarjeta | /tarjetas/{id}/private | GET | Requiere autenticación | Obtiene datos privados de una tarjeta | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear tarjeta | /tarjetas | POST | Requiere autenticación | Crea una nueva tarjeta | 201 Created | 400 Bad Request, 409 Conflict |
| Actualizar tarjeta | /tarjetas/{id} | PUT | Requiere autenticación | Actualiza una tarjeta por su ID | 200 OK | 400 Bad Request, 404 Not Found |
| Eliminar tarjeta | /tarjetas/{id} | DELETE | Requiere autenticación | Elimina una tarjeta por su ID | 204 No Content | 404 Not Found |

Usuarios

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener todos los usuarios | /usuarios | GET | Requiere autenticación | Obtiene todos los usuarios paginados | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener usuario por ID | /usuarios/{id} | GET | Requiere autenticación | Obtiene un usuario por su ID | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Crear usuario | /usuarios | POST | Requiere autenticación | Crea un nuevo usuario | 201 Created | 400 Bad Request, 409 Conflict |
| Actualizar usuario | /usuarios/{id} | PUT | Requiere autenticación | Actualiza un usuario por su ID | 200 OK | 400 Bad Request, 404 Not Found |
| Eliminar usuario | /usuarios/{id} | DELETE | Requiere autenticación | Elimina un usuario por su ID | 204 No Content | 404 Not Found |
| Registrar usuario ya creado | /auth/signin | POST | No requiere autenticación | Registra un usuario que ya ha sido creado anteriormente | 200 OK | 400 Bad Request,  404 Not Found |
| Registrar usuario | /auth/signup | POST | No requiere autenticación | Registra un usuario nuevo | 200 OK | 400 Bad Request |

Divisas

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Obtener divisas | /divisas/latest | GET | Requiere autenticación | Obtiene el intercambio de divisas | 200 OK | 401 Unauthorized, 403 Forbidden |

Storage

| Endpoint | URL | HTTP Verbo | AUTH | Descripción | HTTP Status Code | Otras Salidas |
| --- | --- | --- | --- | --- | --- | --- |
| Exportar Zip | /storage/zip/generate | POST | Requiere autenticación | Exporta el archivo .zip de la copia de seguridad | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Importar Zip | /storage/zip/import/{filename.+} | POST | Requiere autenticación | Importar el archivo .zip de la copia de seguridad | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Json Clientes | /storage/json/generate/{guid} | POST | Requiere autenticación | Exporta el archivo .json de cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Json Clientes Admin | /storage/jsonClientesAdmin/generate | POST | Requiere autenticación | Exporta el archivo .json de clientes para los admin | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Json Movimientos Cliente | /storage/jsonMovimientos/generate/{id} | POST | Requiere autenticación | Exporta el archivo .json de movimientos del cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Json Movimientos Admin | /storage/jsonMovimientos/generate | POST | Requiere autenticación | Exporta el archivo .json de todos los movimientos de todos los clientes | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Pdf Movimientos Clientes | /storage/pdfMovimientos/generate | POST | Requiere autenticación | Exporta el archivo .pdf de todos los movimientos de todos los clientes | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Exportar Pdf Movimientos Cliente | /storage/pdfMovimientos/generate/{guid} | POST | Requiere autenticación | Exporta el archivo .pdf de movimientos del cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Importar Csv | /storage/csvProductos/import | POST | Requiere autenticación | Importar el archivo .csv de productos | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Zip | /storage/zip/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .zip de la copia de seguridad | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Json Clientes | /storage/jsonClientes/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .json de clientes | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Json Cliente | /storage/jsonClientes/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .json de cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Json Movimientos | /storage/jsonMovimientos/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .json de movimientos | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Json Movimientos Cliente | /storage/jsonMovimientos/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .json de movimientos de un cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Pdf Movimientos | /storage/pdfMovimientos/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .pdf de movimientos | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |
| Obtener Pdf Movimientos Cliente | /storage/pdfMovimientos/{filename.} | GET | Requiere autenticación | Obtiene el contenido del archivo .pdf de movimientos de un cliente | 200 OK | 401 Unauthorized, 403 Forbidden, 404 Not Found |

**Tests:**

### **Pruebas en el Desarrollo de Software**

En el ámbito del desarrollo de software, las pruebas son esenciales para garantizar la calidad y fiabilidad de un proyecto. Estas pruebas permiten comprobar que cada componente del sistema funciona correctamente, ya sea de forma aislada o en interacción con otros. Implementar pruebas desde las primeras etapas del desarrollo ayuda a identificar errores tempranamente, reduciendo el esfuerzo necesario para solucionarlos más adelante. Además, aseguran que los cambios futuros, como la implementación de nuevas funcionalidades o la refactorización de código, no afecten el comportamiento esperado del sistema.

Existen diversos tipos de pruebas, entre los que destacan:

* Pruebas unitarias: Validan el funcionamiento de métodos o funciones específicas.
* Pruebas de integración: Evalúan la interacción entre distintos módulos o componentes del sistema.
* Pruebas funcionales: Aseguran que la aplicación cumple con los requisitos establecidos por los usuarios finales.

### **Herramientas Utilizadas: JUnit y Mockito**

En nuestro proyecto, empleamos JUnit y Mockito como herramientas clave para la ejecución de pruebas:

* JUnit: Una biblioteca ampliamente utilizada para pruebas unitarias en Java. JUnit nos permite definir casos de prueba claros y automatizados, verificando el comportamiento de métodos individuales. Gracias a esta herramienta, podemos garantizar que cada unidad de código funcione correctamente, incluso tras realizar cambios o mejoras. Los resultados de las pruebas son rápidos, fáciles de interpretar y fundamentales para mantener la calidad del proyecto.
* Mockito: Para manejar dependencias externas, como bases de datos o servicios remotos, usamos Mockito. Esta herramienta nos permite simular dependencias y centrarnos únicamente en la funcionalidad que estamos probando. Con Mockito, conseguimos aislar los componentes bajo prueba, asegurando que los resultados sean fiables y las pruebas más eficientes.

### **Beneficios de las Pruebas en Nuestro Proyecto**

El uso de JUnit y Mockito ha aportado múltiples ventajas, entre ellas:

* Confianza en el código: Sabemos que los cambios no introducirán errores, ya que los tests los detectan inmediatamente.
* Identificación temprana de fallos: Las pruebas unitarias ayudan a localizar errores rápidamente durante el desarrollo.
* Facilidad en el mantenimiento: Los tests permiten refactorizar el código con seguridad, garantizando que las funcionalidades existentes sigan operando correctamente.
* Aislamiento de dependencias: Con Mockito, podemos probar módulos sin necesidad de configuraciones complejas o servicios externos.

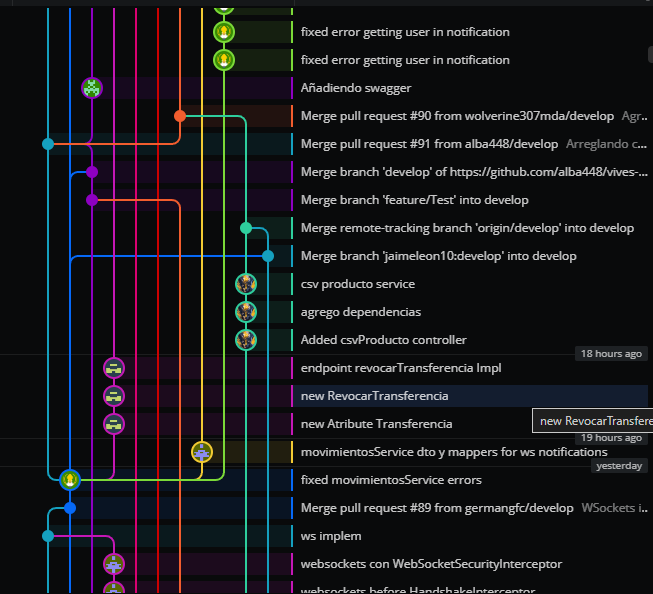
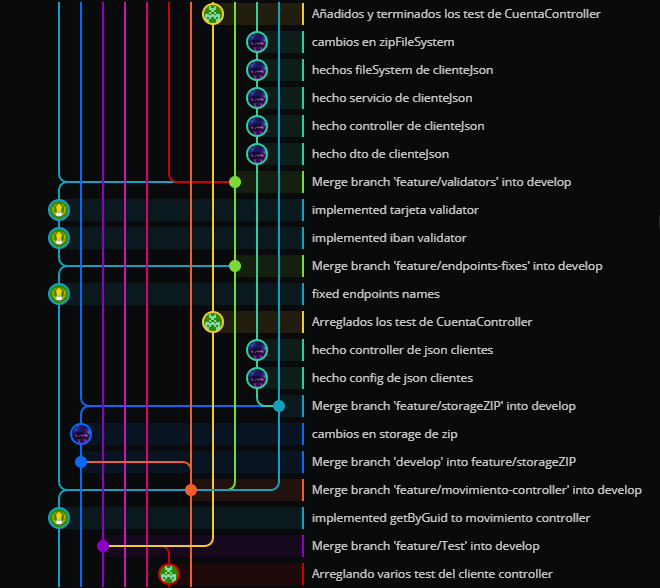
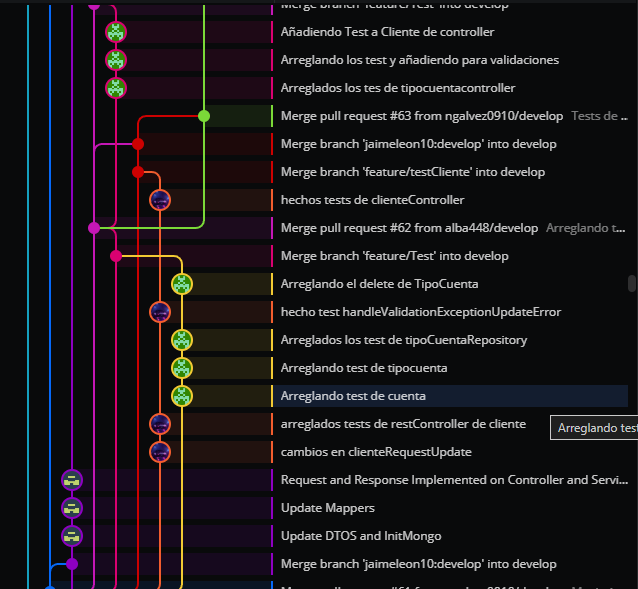
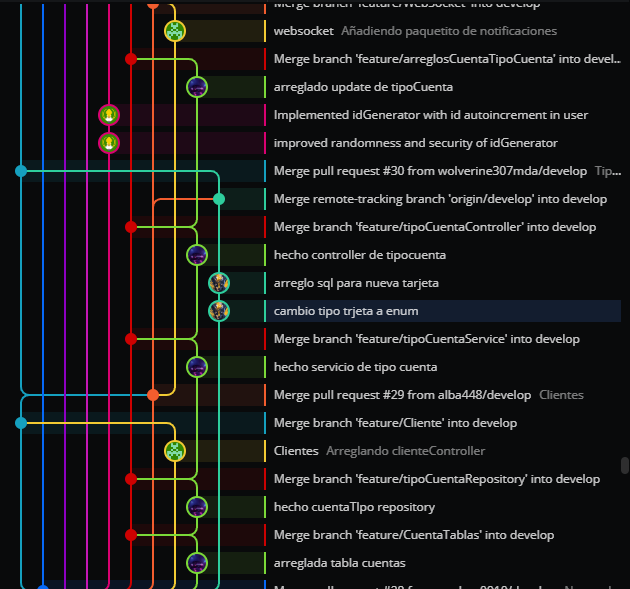
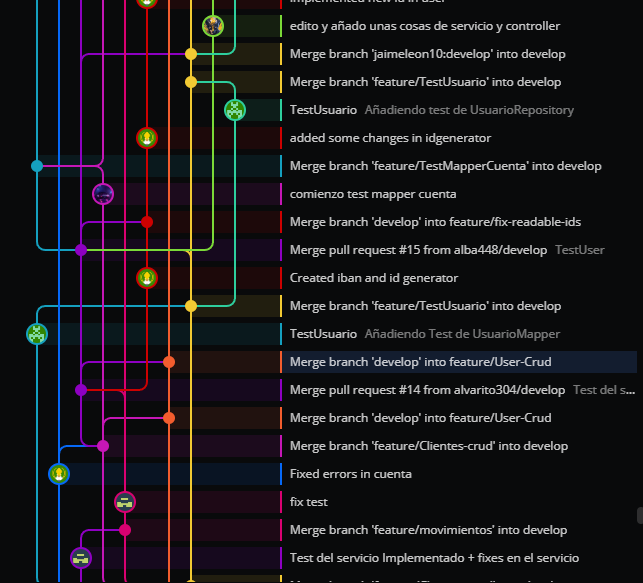
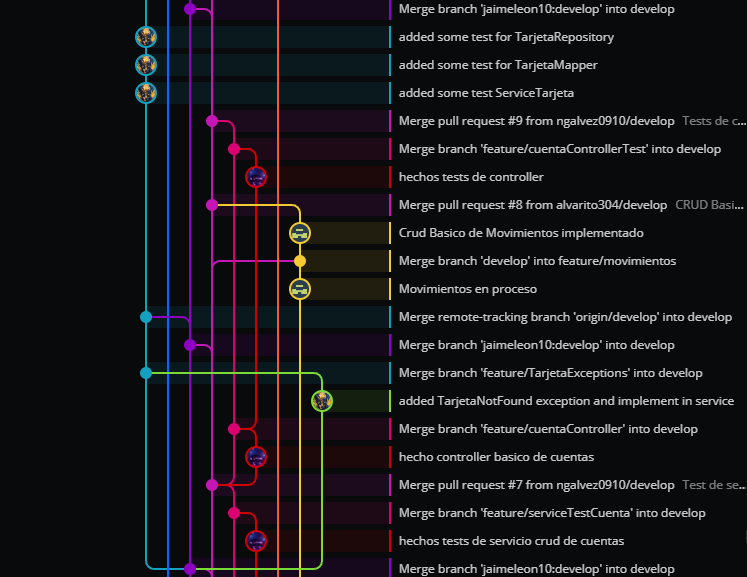
**Organización: Trello**

Trello es una herramienta de organización y gestión de tareas basada en tableros. Puedes usarla para organizar proyectos de manera visual mediante tarjetas que representan tareas y listas que agrupan esas tarjetas por categorías o etapas (como "Por hacer", "En progreso" y "Completado"). Cada tarjeta puede incluir detalles como descripciones, checklists, fechas de vencimiento, comentarios y archivos adjuntos. Es súper útil para trabajos en equipo porque todos pueden ver los avances y colaborar en tiempo real. Además, es fácil de usar y se adapta a cualquier tipo de proyecto, desde estudios hasta proyectos más grandes.

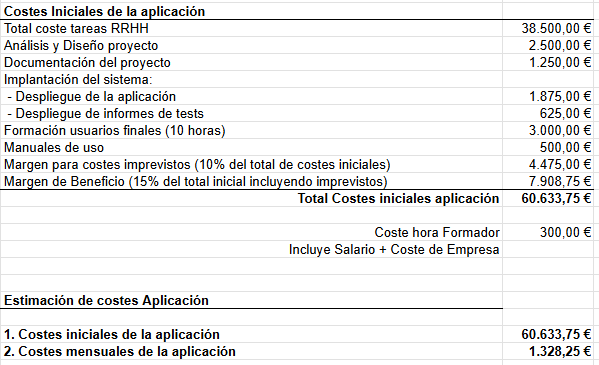
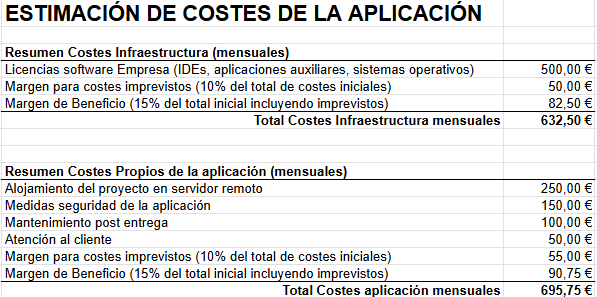
**GitFlow**

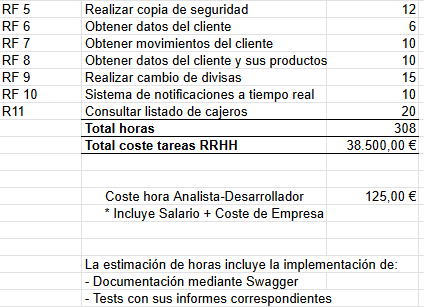
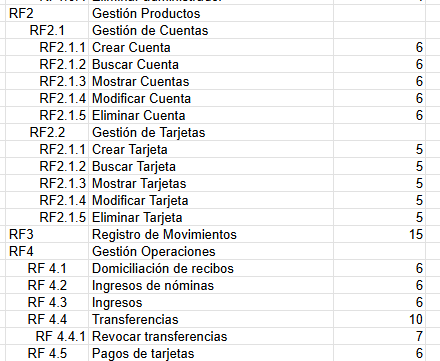
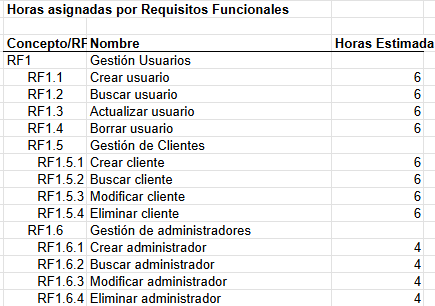
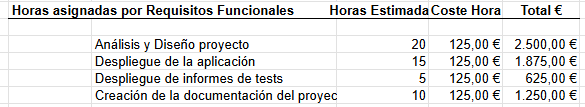
GitFlow es una metodología que organiza el uso de ramas en un proyecto para trabajar con Git de manera más eficiente, dividiendo el desarrollo en ramas principales como main (producción) y develop (desarrollo), y usando ramas temporales para funcionalidades (feature), corrección de errores (hotfix) y lanzamientos (release). Esta estructura facilita mantener el código limpio y organizado, especialmente en equipos grandes.

Para aprovechar al máximo esta metodología, herramientas como GitKraken son muy útiles. GitKraken es una interfaz gráfica que simplifica la gestión de Git, permitiéndote visualizar de forma clara los commits, ramas y merges. Además, integra soporte para GitFlow, lo que hace que trabajar con esta metodología sea más intuitivo gracias a sus flujos predefinidos y accesibles con solo unos clics. De esta forma, combinar GitFlow y GitKraken mejora la colaboración en equipo y evita complicaciones al manejar ramas y versiones de código.



**Costes**

****

****

**Bibliografía**

Repositorios en GitHub

González Sánchez, J.L. (2024). *Docker-tutorial* [Página web].

<https://github.com/joseluisgs/docker-tutorial>

González Sánchez, J.L. (2024). *DesarrolloWebEntornosServidor-02-2024-2025* [Página web].

<https://github.com/joseluisgs/DesarrolloWebEntornosServidor-02-2024-2025>

Páginas web

Rishi. (2023). *Implementing Redis in Spring Boot* [Página web].

<https://awstip.com/implementing-redis-in-spring-boot-3d2756e5ab69>

Gitignore [Página web].

<https://www.toptal.com/developers/gitignore>

ApiRest

Frankfurter API. (n.d.). *The frankfurter API for foreign exchange rates* [Api Rest].

<https://frankfurter.dev/>