Vives Bank

2º DAW IES LUIS VIVES

Jaime León Mulero

Yahya el Hadri el Bakkali

Javier Ruiz Serrano

Alba García Orduña

Natalia González Álvarez

Mario de Domingo Álvarez

Óscar Encabo Nieto

Kelvin Sánchez

ÍNDICE

1. 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.

2. 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.

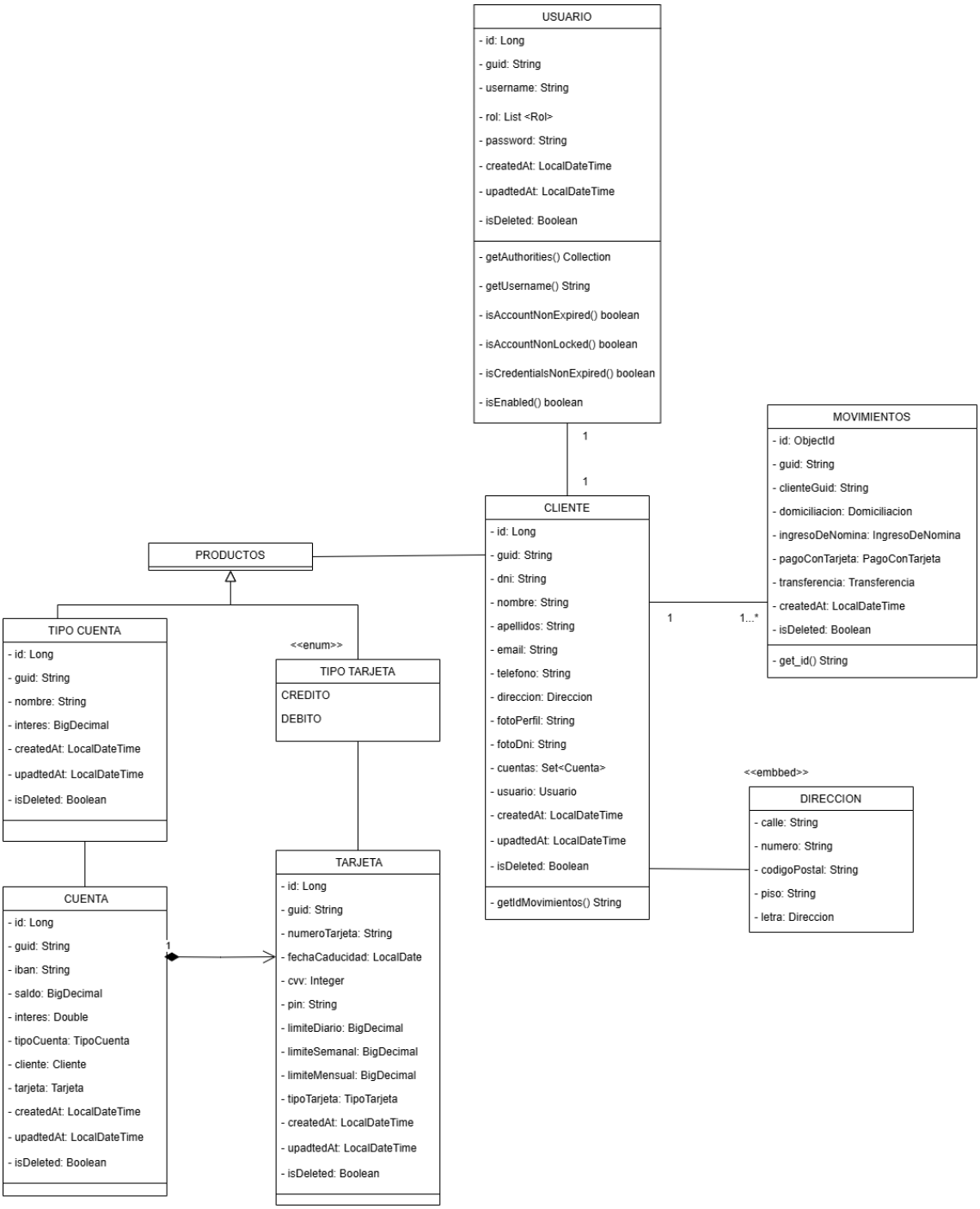
RI 13.- El cliente puede contratar una cuenta que puede ser normal, cuyo interés es del 2% o de ahorro cuyo interés es del 1%.

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.

3. Diagrama

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.



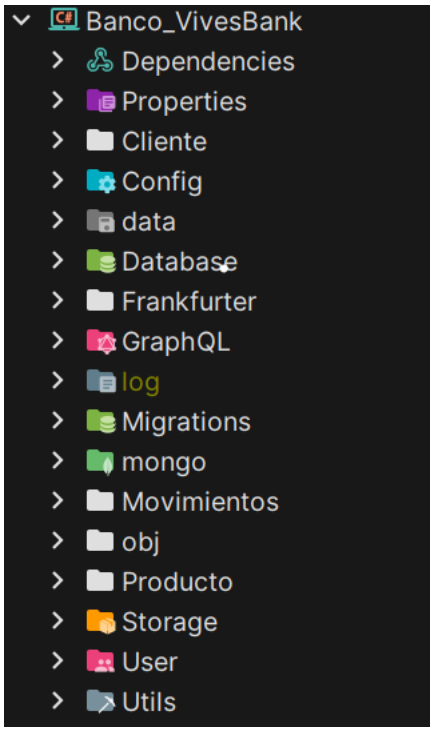
4. Funcionamiento del Programa

Tecnologías, enfoques y lenguajes usados

- Arquitectura orientada al dominio.
- Manejo de excepciones orientadas al dominio.
- Uso de inyección de dependencias.
- C#
- Docker
- PostgreSQL
- MongoDB
- Postman
- Retrofit
- Git
- GitFlow
- NuGet
- Serilog
- Swagger
- NUnit
- Moq
- TestContainers

Estructura del Proyecto Banco_VivesBank

El proyecto sigue una arquitectura modular y está organizado en carpetas que reflejan los distintos dominios del sistema bancario. Esto facilita la escalabilidad y el mantenimiento del código.



🗁 Cliente

En esta carpeta se gestiona todo lo relacionado con los clientes del banco. Aquí se encuentran los controladores, servicios encargados de manejar la información personal de los clientes, como su nombre, dirección, número de identificación y otros datos relevantes.

ClienteController.cs → Define los endpoints de la API REST para gestionar los clientes (registro, actualización, eliminación, consulta).

ClienteService.cs → Contiene la lógica de negocio para procesar información de los clientes.

📂 Producto

Este módulo se encarga de gestionar los productos financieros del banco, como cuentas bancarias, tarjetas y productos base. Se divide a su vez en:

🏦 *Cuenta*

Maneja toda la información relacionada con las cuentas bancarias de los clientes, incluyendo saldos y operaciones.

- CuentaController.cs → API REST para gestionar cuentas bancarias.
- CuentaService.cs → Lógica de negocio para cuentas.
- CuentaRepository.cs → Acceso a la base de datos.

💳 *Tarjeta*

Administra las tarjetas de débito y crédito de los clientes, permitiendo activación, bloqueo y transacciones.

- TarjetaController.cs → API para gestión de tarjetas.
- TarjetaService.cs → Lógica de negocio de las tarjetas.

📒 *ProductoBase*

Aquí se almacenan definiciones de los productos financieros del banco, incluyendo características y condiciones.

- ProductoBaseController.cs → Controlador de productos financieros.
- ProductoBaseService.cs → Lógica de gestión de productos base.

📂 Frankfurter

Este módulo está relacionado con la API externa Frankfurter, utilizada para realizar conversiones de divisas en tiempo real.

FrankfurterService.cs → Se encarga de realizar peticiones a la API de Frankfurter y procesar los datos recibidos.

📂 Movimientos

Este paquete gestiona todas las operaciones financieras de los clientes, como transferencias, depósitos, pagos con tarjeta y domiciliaciones.

MovimientoController.cs → Define los endpoints para registrar y consultar movimientos bancarios.

MovimientoService.cs → Contiene la lógica de negocio para procesar transacciones.

🗁 Storage

Módulo encargado del almacenamiento de archivos en el sistema. Se encarga de la generación y gestión de documentos en diferentes formatos como JSON, PDF y CSV, así como de la creación de copias de seguridad en formato ZIP.

FileStorageService.cs → Maneja la lectura, escritura y eliminación de archivos almacenados.

🗁 Users

Aquí se gestionan los usuarios del sistema, tanto clientes como administradores del banco.

UserController.cs → Controlador para la administración de usuarios.

UserService.cs → Contiene la lógica de negocio relacionada con la gestión de usuarios.

🗁 Utils

Paquete que agrupa herramientas y funciones auxiliares que pueden ser utilizadas en todo el proyecto.

🔑 *Security.Auth*

Este módulo se encarga de la autenticación y seguridad del sistema. Aquí se manejan los procesos de inicio de sesión, generación y validación de tokens JWT, así como la gestión de credenciales.

Jwt.cs → Implementa funciones auxiliares para la gestión de tokens JWT.

🛠 *Generators*

Esta carpeta contiene clases especializadas en la generación automática de valores esenciales dentro del sistema bancario, como números de cuenta, códigos CVV y fechas de expiración de tarjetas.

CvvGenerator.cs → Genera códigos de seguridad (CVV) para tarjetas bancarias de forma aleatoria y segura.

ExpDateGenerator.cs → Crea fechas de expiración para tarjetas de crédito o débito, asegurando su validez.

GuidGenerator.cs → Genera identificadores únicos universales (UUID/GUID) para distintos elementos dentro del sistema.

IbanGenerator.cs → Genera números de cuenta bancaria en formato IBAN, asegurando que sean válidos según las normas bancarias.

TarjetaGenerator.cs → Crea números de tarjetas de crédito o débito siguiendo los estándares internacionales (como el algoritmo de Luhn).

Estas clases se utilizan para la creación automática de datos en pruebas o procesos internos del sistema.

🕮Carpeta Pagination

Aquí se encuentran las clases encargadas de manejar la paginación de resultados en las consultas a la base de datos, asegurando eficiencia en la carga y visualización de datos en la aplicación.

✓Carpeta Validators

Esta carpeta agrupa validadores para verificar la integridad y validez de datos bancarios, como tarjetas, cuentas bancarias y documentos de identidad.

CardLimitValidators.cs → Valida los límites de crédito o saldo disponibles en las tarjetas bancarias.

CreditCardValidation.cs → Implementa validaciones para números de tarjeta de crédito utilizando el algoritmo de Luhn y otros controles.

DniValidation.cs → Verifica la validez de documentos de identidad (DNI, NIF, etc.), asegurando que cumplan con las reglas del país correspondiente.

IbanValidator.cs → Comprueba que los números de cuenta IBAN sean correctos, aplicando validaciones de formato y checksum.

🗃 Websocket.Notifications

Este módulo está relacionado con el envío de notificaciones en tiempo real a los usuarios mediante WebSockets. Se usa, por ejemplo, para alertar a los clientes sobre nuevos movimientos en su cuenta.

NotificationService.cs → Implementa el servicio para el envío de notificaciones en tiempo real.

✸ Program.cs

Archivo principal del proyecto que contiene el punto de entrada del programa. Es donde se inicializan los servicios y se ejecuta la aplicación en .NET.

⚙ Configuración

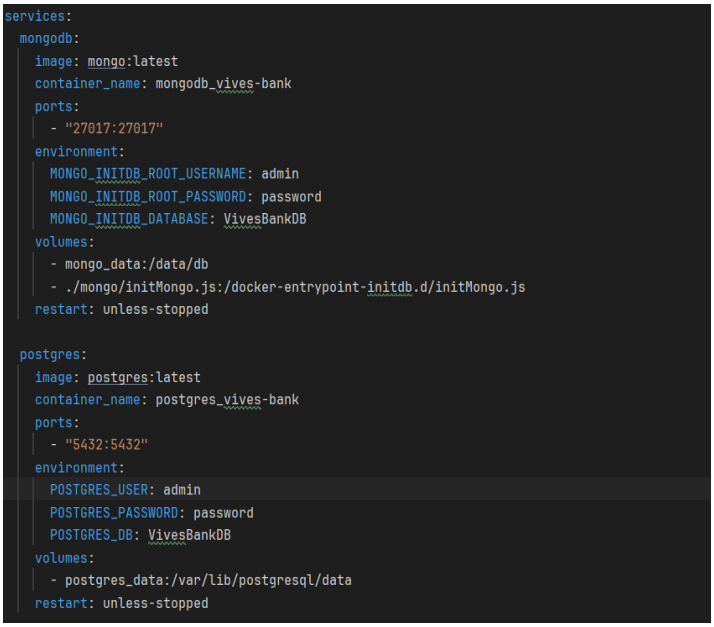
Los archivos de configuración definen propiedades esenciales del proyecto, como conexiones a bases de datos y configuraciones para diferentes entornos (desarrollo y producción).

appsettings.json → Archivo de configuración general del sistema.

appsettings.Development.json → Configuración específica para desarrollo.

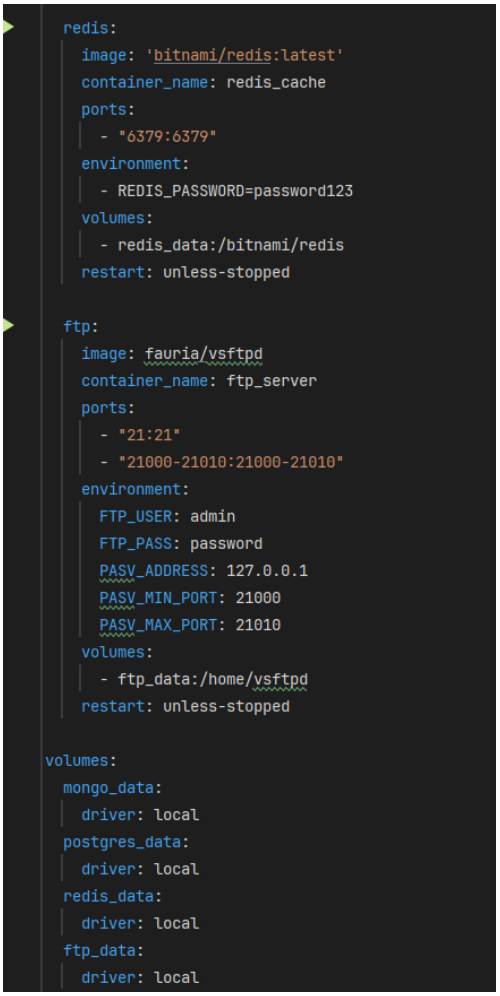
Docker-Compose

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.



```yaml
services:
  mongodb:
    image: mongo:latest
    container_name: mongodb_vives-bank
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: password
      MONGO_INITDB_DATABASE: VivesBankDB
    volumes:
      - mongo_data:/data/db
      - ./mongo/initMongo.js:/docker-entrypoint-initdb.d/initMongo.js
    restart: unless-stopped

  postgres:
    image: postgres:latest
    container_name: postgres_vives-bank
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
      POSTGRES_DB: VivesBankDB
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
```



```yaml
  redis:
    image: 'bitnami/redis:latest'
    container_name: redis_cache
    ports:
      - "6379:6379"
    environment:
      - REDIS_PASSWORD=password123
    volumes:
      - redis_data:/bitnami/redis
    restart: unless-stopped

  ftp:
    image: fauria/vsftpd
    container_name: ftp_server
    ports:
      - "21:21"
      - "21000-21010:21000-21010"
    environment:
      FTP_USER: admin
      FTP_PASS: password
      PASV_ADDRESS: 127.0.0.1
      PASV_MIN_PORT: 21000
      PASV_MAX_PORT: 21010
    volumes:
      - ftp_data:/home/vsftpd
    restart: unless-stopped

volumes:
  mongo_data:
    driver: local
  postgres_data:
    driver: local
  redis_data:
    driver: local
  ftp_data:
    driver: local
```

Configuración de la base de datos PostgreSQL

Uno de los servicios principales es postgres-db, 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.

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 caché en nuestro proyecto

El sistema de caché del proyecto está basado en una combinación de **caché en memoria** y **Redis**, lo que permite aprovechar lo mejor de ambos mundos.

- **Caché en memoria:** Utiliza la memoria local del servidor para almacenar datos de acceso frecuente, lo que garantiza una latencia extremadamente baja y un rendimiento rápido. Este tipo de caché es ideal para almacenar información temporal o datos que no requieren persistencia a largo plazo, ya que se pierde cuando el proceso se detiene o se reinicia.
- **Caché en Redis:** Redis es un sistema de caché distribuido basado en un almacén de datos en memoria de alto rendimiento. Utilizamos la imagen redis:7.4.1 y exponemos su puerto estándar (6379). Redis permite la persistencia de datos, ya que utiliza un volumen denominado redis-data para almacenar información de manera duradera. Esto asegura que, incluso si el contenedor de Redis se detiene o reinicia, los datos almacenados no se perderán.

Esta combinación permite que los datos más frecuentes se almacenen en caché en memoria, proporcionando tiempos de acceso ultra rápidos, mientras que Redis gestiona un caché persistente y distribuido que asegura la fiabilidad de los datos a largo plazo y la escalabilidad horizontal. Además, Redis ofrece características avanzadas como la expiración de claves, la persistencia en disco y la replicación, lo que facilita la gestión de datos de manera robusta y eficiente.

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.

GraphQL

GraphQL es un lenguaje de consulta para APIs que permite a los clientes solicitar exactamente los datos que necesitan, evitando respuestas innecesarias o datos redundantes. A diferencia de **REST**, donde cada endpoint devuelve una estructura fija de datos, **GraphQL** permite obtener múltiples recursos en una sola petición, optimizando el rendimiento y la flexibilidad de la API.

🚀 *Implementación en el Servicio de Movimientos*

En Banco_VivesBank, hemos implementado GraphQL en el servicio de movimientos para gestionar de manera eficiente las transacciones bancarias de los clientes. Dado que los movimientos pueden incluir una gran cantidad de información (como transacciones, pagos, ingresos y transferencias), GraphQL nos permite optimizar la recuperación de datos y adaptarla a las necesidades específicas de cada consulta.

- Consulta personalizada: Los clientes pueden solicitar solo los movimientos que les interesan, como los de un período específico o los relacionados con una cuenta en particular.

- Relación con MongoDB: Dado que los movimientos se almacenan en MongoDB, GraphQL facilita la consulta de documentos sin necesidad de múltiples endpoints.

- Mejora del rendimiento: Se reduce la cantidad de datos transmitidos en la red, optimizando la velocidad de respuesta y la experiencia del usuario.

✅ *Beneficios de Usar GraphQL en Movimientos*

✓ Consulta de datos eficiente: Permite a los clientes solicitar solo los campos que necesitan, evitando la sobrecarga de datos innecesarios.
✓ Optimización del rendimiento: Al reducir las llamadas al servidor y la cantidad de datos transferidos, la API responde más rápido.
✓ Flexibilidad en la API: No es necesario crear múltiples endpoints para cada tipo de consulta; una sola entrada puede manejar todas las peticiones.
✓ Evolución sin romper compatibilidad: Se pueden agregar nuevos campos o relaciones sin afectar a los clientes existentes.

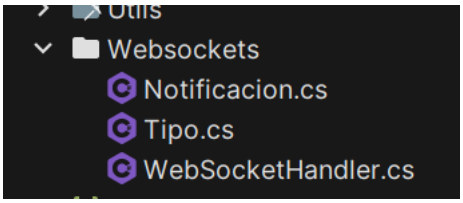
ApiRest

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.

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.

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.

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 Guid	/clientes/{guid}	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
MeActualizar	/clientes	PUT	Requiere autenticación	Actualiza un cliente por su ID	200 OK	400 Bad Request, 404 Not Found
Eliminar cliente	/clientes/{guid}	DELETE	Requiere autenticación	Elimina un cliente por su ID	204 No Content	404 Not Found
MeProfile	/clientes/me	GET	Requiere autenticación	Obtiene el perfil del cliente	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
MeDelete	/clientes	DELETE	Requiere autenticación	Elimina el perfil del cliente	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found

UpdateMyProfilePicture	/clientes/fotoPerfil	PUT	Requiere autenticación	Actualiza la foto de perfil	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
UpdateMyDniPicture	/clientes/fotoDni	PUT	Requiere autenticación	Actualiza la foto del Dni	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
GetFotoDni	/clientes/fotoDni{guid}	GET	Requiere autenticación	Obtiene la foto del Dni	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
GetMovimiento	/clientes/movimientosPDF	GET	Requiere autenticación	Obtiene los movimientos	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 guid	/cuentas/{guid}	GET	Requiere autenticación	Obtiene una cuenta por su ID	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
Obtener todas por guid del cliente	/cuentas/cliente/{guid}	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/{ guid }	PUT	Requiere autenticación	Actualiza una cuenta por su ID	200 OK	400 Bad Request, 404 Not Found
Eliminar cuenta	/cuentas/{guid}	DELETE	Requiere autenticación	Elimina una cuenta por su ID	204 No Content	404 Not Found
GetByIban	/cuentas/admin/iban{iban}	GET	Requiere autenticación	Obtiene la cuenta mediante su Iban	200 OK	400 Bad Request, 404 Not Found
DeleteAdmin	/cuentas/admin {guid}	DELETE	Requiere autenticación	Un admin desactiva la cuenta	200 OK	400 Bad Request, 404 Not Found
GetAllMeAccounts	/cuentas/me	GET	Requiere autenticación	Muestra todas las cuentas del cliente	200 OK	400 Bad Request, 404 Not Found
GetMeByIban	/cuentas/iban/ {iban}	GET	Requiere autenticación	Muestra la cuenta del cliente por su iban	200 OK	400 Bad Request, 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 GUID	/movimientos /{guid}	GET	Requiere autenticación	Obtiene un movimiento por su GUID	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
Obtener movimientos por el GUID del cliente	/movimientos/cliente/{clienteguid}	GET	Requiere autenticación	Obtiene movimientos por el GUID del cliente	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found

GetMyMovimiento	/movimientos/me	GET	Requiere autenticación	Le muestra todos los movimiento al cliente	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
RevocarTransferencia	/movimientos/transferencia/revocar/{movimientoGuid}	DELETE	Requiere autenticación	Cancela un movimiento	200 OK	400 Bad Request, 401 Unauthorized, 403 Forbidden

Domiciliaciones

Endpoint	URL	HTTP Verbo	AUTH	Descripción	HTTP Status Code	Otras Salidas
Obtener todos los movimientos	/domiciliaciones	GET	Requiere autenticación	Obtiene todos las domiciliaciones	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
Obtener una domiciliacion por su GUID	/domiciliaciones/{domiciliacionesguid}	GET	Requiere autenticación	Obtiene una domiciliacion por su GUID	200 OK	401 Unauthorized, 403 Forbidden,

						404 Not Found
Obtener domiciliacion por el GUID del cliente	/domiciliaciones/cliente/{clienteguid}	GET	Requiere autenticación	Obtiene domiciliacion por el GUID del cliente	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
GetMy Domiciliacion	/domiciliaciones/me	GET	Requiere autenticación	Le muestra todos las domiciliaciones al cliente	200 OK	400 Bad Request, 401 Unauthorized, 403 Forbidden
Crear domiciliacion	/domiciliaciones	POST	Requiere autenticación	Crea una domiciliacion	200 OK	400 Bad Request, 401 Unauthorized, 403 Forbidden
DesactivateDomiciliacion	/domiciliaciones/{domiciliacionGuid}	DELETE	Requiere autenticación	Cancela una domiciliacion el Admin	200 OK	400 Bad Request, 401 Unauthorized, 403 Forbidden
DesactivateMyDomiciliacion	/domiciliaciones/me/{domiciliacionGuid}	DELETE	Requiere autenticación	Cancela una domiciliacion el propio cliente	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 GUID	/tarjetas/{guid}	GET	Requiere autenticación	Obtiene una tarjeta por su GUID	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
GetTarjetaByNumeroTarjeta	/tarjetas/numero/{numeroTarjeta}	GET	Requiere autenticación	Obtiene la tarjeta por su numero	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/{guid}	PUT	Requiere autenticación	Actualiza una tarjeta por su GUID	200 OK	400 Bad Request, 404 Not Found
Eliminar tarjeta	/tarjetas/{guid}	DELETE	Requiere autenticación	Elimina una tarjeta por su GUID	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 GUID	/usuarios/{guid}	GET	Requiere autenticación	Obtiene un usuario por su GUID	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found
GetMe	/usuarios/me	GET	Requiere autenticación	Obtiene los datos del usuarios	200 OK	401 Unauthorized, 403 Forbidden, 404 Not Found

GetByUsername	/usuarios/username/{username}	GET	Requiere autenticación	Obtiene el usuario por username	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/{guid}	PUT	Requiere autenticación	Actualiza un usuario por su ID	200 OK	400 Bad Request, 404 Not Found
UpdateMyPassword	/usuarios/password	PUT	Requiere autenticación	El usuario actualiza su propia contraseña	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

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	200 OK	401 Unauthorized, 403 Forbidden,

				copia de seguridad		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: NUnit y Mock

En nuestro proyecto, utilizamos **NUnit** y **Moq** como herramientas clave para la ejecución de pruebas:

- **NUnit:** Es una de las bibliotecas más populares para pruebas unitarias en C#. Con NUnit, podemos definir y ejecutar casos de prueba de manera sencilla y efectiva. Nos permite verificar el comportamiento de los métodos individuales, asegurando que cada unidad de código funcione correctamente. Además, proporciona un entorno adecuado para la automatización de pruebas, lo que facilita la integración continua. Gracias a esta herramienta, podemos tener resultados rápidos y fácilmente interpretables, fundamentales para mantener la calidad y estabilidad de nuestro proyecto.
- **Moq:** Para manejar dependencias externas, como bases de datos o servicios remotos, usamos Moq. Esta herramienta nos permite crear **mocks** de las dependencias, simulando su comportamiento de forma controlada. Esto nos permite centrarnos únicamente en la funcionalidad del componente bajo prueba, sin necesidad de involucrar servicios o recursos externos. Moq hace que nuestras pruebas sean más rápidas y confiables, al aislar el código bajo prueba y evitar que interacciones externas interfieran en los resultados.

Beneficios de las Pruebas en Nuestro Proyecto

El uso de **NUnit** y **Moq** nos ha proporcionado múltiples ventajas, entre ellas:

- **Confianza en el código:** Sabemos que los cambios no introducirán errores imprevistos, ya que los **tests** se ejecutan automáticamente y detectan cualquier fallo de manera inmediata.
- **Identificación temprana de fallos:** Las pruebas unitarias nos permiten detectar errores rápidamente durante el desarrollo, evitando que se acumulen problemas en etapas posteriores del ciclo de vida del proyecto.
- **Facilidad en el mantenimiento:** Gracias a los **tests**, podemos refactorizar el código con seguridad, garantizando que las funcionalidades existentes sigan operando correctamente sin afectar el comportamiento general de la aplicación.
- **Aislamiento de dependencias:** Con **Moq**, podemos simular dependencias externas y pruebas de componentes individuales sin necesidad de configurar bases de datos o servicios externos. Esto nos permite ejecutar pruebas más rápidas, confiables y fáciles de mantener.

Uso de TestContainer

En el desarrollo del proyecto Banco_VivesBank, se ha implementado Testcontainers para realizar pruebas integradas de los servicios de cada dominio. Gracias a esta herramienta, es posible ejecutar bases de datos y otros servicios en contenedores Docker, asegurando que las pruebas se realicen en un entorno controlado y lo más parecido posible a producción.

🚀*Uso en los Servicios del Proyecto*

Cada servicio dentro del sistema ha sido testeado mediante Testcontainers, asegurando que la lógica de negocio y la comunicación con las bases de datos funcionen correctamente antes de desplegar la aplicación.

Servicios con Base de Datos Relacional (PostgreSQL):

- ClienteService
- CuentaService
- TarjetaService
- ProductoBaseService

Estos servicios utilizan PostgreSQL dentro de un contenedor para probar todas las operaciones de persistencia, asegurando la integridad de los datos y el correcto funcionamiento de las consultas SQL.

Servicio con Base de Datos NoSQL (MongoDB):

- MovimientoService

Como los movimientos bancarios requieren una base de datos más flexible y escalable, se ha optado por MongoDB, que permite un almacenamiento eficiente de grandes volúmenes de transacciones. Con Testcontainers, se levanta un contenedor de MongoDB en cada prueba para garantizar la compatibilidad del servicio.

✅*Beneficios de Usar Testcontainers*

- ✓ Pruebas más realistas: Se ejecutan en un entorno similar a producción, evitando problemas de configuración entre entornos.

- ✓ Independencia del entorno local: No es necesario instalar PostgreSQL o MongoDB en el equipo de desarrollo; los tests se ejecutan en contenedores aislados.

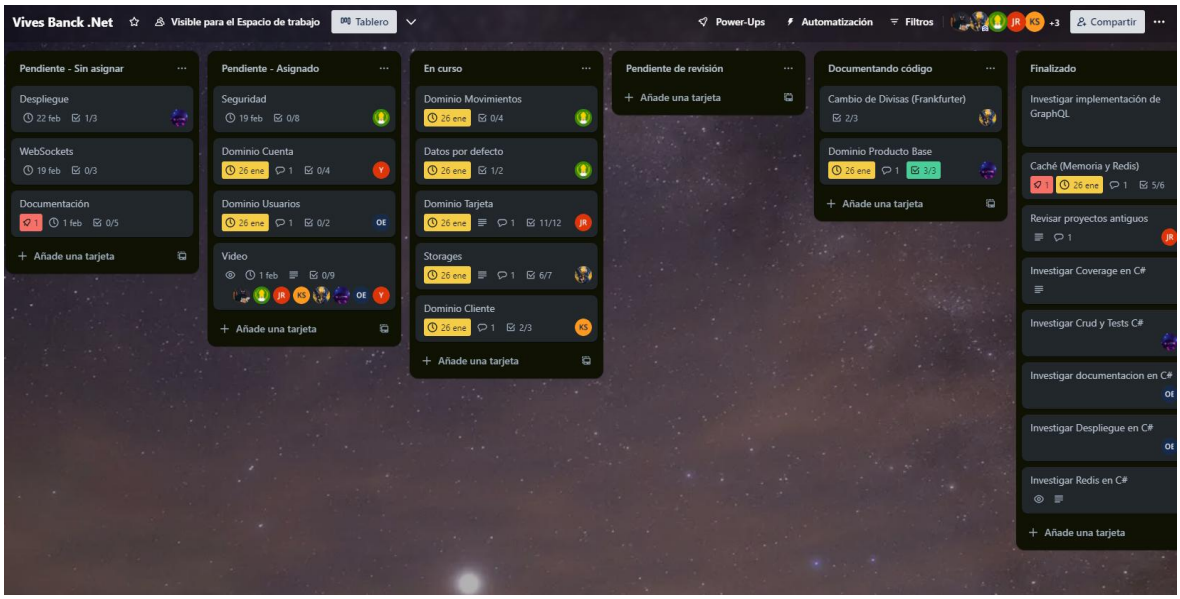
- ✓ Automatización y facilidad de integración: Se pueden integrar fácilmente con frameworks de pruebas como xUnit o NUnit en .NET.

- ✓ Limpieza automática: Una vez finalizadas las pruebas, los contenedores se eliminan, asegurando que cada test se ejecute en un entorno limpio.

- ✓ **Flexibilidad para pruebas de integración**: Permite probar interacciones entre múltiples servicios sin necesidad de mockear bases de datos.

Organización: Trello

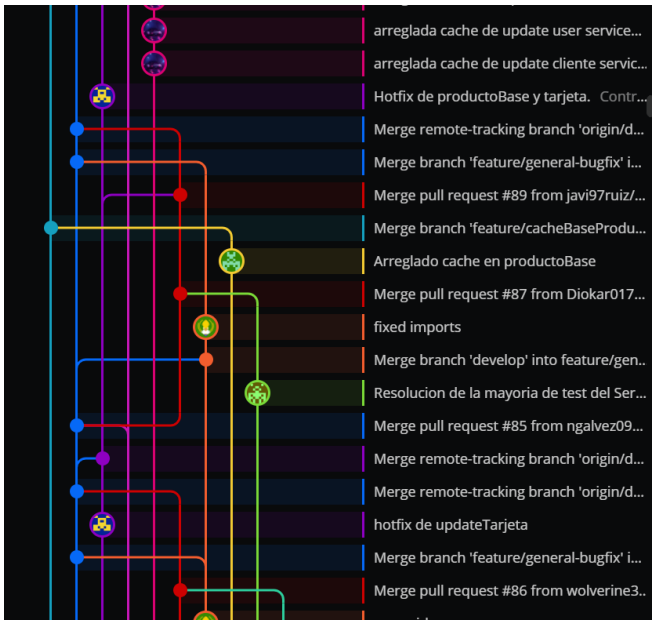
Trello es una herramienta de organización y gestión de tareas basada en tableros. Puedes usarla para organizar proyectos de manera visual mediante tarjetas que representan tareas y listas que agrupan esas tarjetas por categorías o etapas (como "Pendiente", "En curso" y "Finalizado"). Cada tarjeta puede 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

ESTIMACIÓN DE COSTES DE LA APLICACIÓN	
Resumen Costes Infraestructura (mensuales)	
Licencias software Empresa (IDEs, aplicaciones auxiliares, sistemas operativos)	1.500,00 €
Margen para costes imprevistos (10% del total de costes iniciales)	150,00 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	412,50 €
Total Costes Infraestructura mensuales	**2.062,50 €**
Resumen Costes Propios de la aplicación (mensuales)	
Alojamiento del proyecto en servidor remoto	500,00 €
Medidas seguridad de la aplicación	250,00 €
Mantenimiento post entrega	350,00 €
Atención al cliente	200,00 €
Margen para costes imprevistos (10% del total de costes iniciales)	130,00 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	214,50 €
Total Costes aplicación mensuales	**1.644,50 €**
Costes Iniciales de la aplicación	
Total coste tareas RRHH	46.750,00 €
Análisis y Diseño proyecto	2.500,00 €
Documentación del proyecto	1.250,00 €
Implantación del sistema:	
- Despliegue de la aplicación	1.875,00 €
- Despliegue de informes de tests	625,00 €
Formación usuarios finales (20 horas)	20.000,00 €
Manuales de uso	1.000,00 €
Margen para costes imprevistos (10% del total de costes iniciales)	5.300,00 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	11.895,00 €
Total Costes iniciales aplicación	**91.195,00 €**
Coste hora Formador	1.000,00 €
Incluye Salario + Coste de Empresa	
Estimación de costes Aplicación	
1. Costes iniciales de la aplicación	**91.195,00 €**
2. Costes mensuales de la aplicación	**3.707,00 €**

Horas asignadas por Requisitos Funcionales	Horas Estimada	Coste Hora	Total €
Análisis y Diseño proyecto	20	125,00 €	2.500,00 €
Despliegue de la aplicación	20	125,00 €	2.500,00 €
Despliegue de informes de tests	10	125,00 €	1.250,00 €
Creación de la documentación del proyec	11	125,00 €	1.375,00 €

Horas asignadas por Requisitos Funcionales

Concepto/RF	Nombre	Horas Estimadas
RF1	Gestión Usuarios	
RF1.1	Crear usuario	6
RF1.2	Buscar usuario	8
RF1.3	Actualizar usuario	8
RF1.4	Borrar usuario	8
RF1.5	Gestión de Clientes	
RF1.5.1	Crear cliente	8
RF1.5.2	Buscar cliente	8
RF1.5.3	Modificar cliente	8
RF1.5.4	Eliminar cliente	8
RF1.6	Gestión de administradores	
RF1.6.1	Crear administrador	7
RF1.6.2	Buscar administrador	6
RF1.6.3	Modificar administrador	6
RF1.6.4	Eliminar administrador	6
RF2	Gestión Productos	
RF2.1	Gestión de Cuentas	
RF2.1.1	Crear Cuenta	10
RF2.1.2	Buscar Cuenta	8
RF2.1.3	Mostrar Cuentas	8
RF2.1.4	Modificar Cuenta	8
RF2.1.5	Eliminar Cuenta	8
RF2.2	Gestión de Tarjetas	
RF2.1.1	Crear Tarjeta	8
RF2.1.2	Buscar Tarjeta	8
RF2.1.3	Mostrar Tarjetas	8
RF2.1.4	Modificar Tarjeta	8
RF2.1.5	Eliminar Tarjeta	8
RF3	Registro de Movimientos	25
RF4	Gestión Operaciones	
RF 4.1	Domiciliación de recibos	8
RF 4.2	Ingresos de nóminas	8
RF 4.3	Ingresos	8
RF 4.4	Transferencias	10
RF 4.4.1	Revocar transferencias	7
RF 4.5	Pagos de tarjetas	6
RF 5	Realizar copia de seguridad	12
RF 6	Obtener datos del cliente	6
RF 7	Obtener movimientos del cliente	10
RF 8	Obtener datos del cliente y sus productos	10
RF 9	Realizar cambio de divisas	15
RF 10	Sistema de notificaciones a tiempo real	10
R11	Consultar listado de cajeros	20
	Total horas	385
	Total coste tareas RRHH	48.125,00 €

Coste hora Analista-Desarrollador	125,00 €
* Incluye Salario + Coste de Empresa	

La estimación de horas incluye la implementación de:
- Documentación mediante Swagger
- Tests con sus informes correspondientes

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-03-2024-2025* [Página web].

https://github.com/joseluisgs/DesarrolloWebEntornosServidor-03-2024-2025

Páginas web

Redis. (2023). *NRedisStack guide (C#/.NET)* [Página web].

https://redis.io/docs/latest/develop/clients/dotnet/

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/