VivesBank.net

By TacoMaxi dev.

Germán Fernández.
Raúl Fernández.
Javier Hernández.
Samuel Cortés.
Alvaro Herrero.
Diego Novillo.
Tomas Vaquerin.

Índice

1- Introducción.

En esta práctica desarrollaremos un sistema para **gestionar los servicios de un banco**. El objetivo principal será registrar a los usuarios, los productos que estos contratan, así como los movimientos que realizan en sus cuentas. Este sistema nos permitirá gestionar eficientemente la información bancaria y ofrecer una experiencia más ordenada y personalizada tanto para los clientes como para el administrador del sistema.

Es importante considerar que un cliente del banco puede disponer de una o varias cuentas. Cada una de estas cuentas estará vinculada exclusivamente a una tarjeta bancaria. Además, el sistema contará con un administrador, cuya función será acceder a toda la información relacionada con los clientes, las cuentas que poseen y los movimientos realizados. Este administrador tendrá privilegios especiales para gestionar y supervisar las operaciones del banco, asegurando un control total sobre las actividades.

Para llevar a cabo este proyecto, utilizaremos una combinación de tecnologías que incluyen una base de datos relacional, como PostgreSQL, y una base de datos NoSQL, como MongoDB. La base de datos relacional nos permitirá organizar la información estructurada, como clientes, cuentas y relaciones entre ellos, mientras que la base de datos NoSQL será útil para almacenar información no estructurada o documentos que complementen nuestro modelo de datos.

El diseño del sistema estará basado en una arquitectura orientada al dominio (DDD - Domain-Driven Design), lo que significa que centraremos nuestro desarrollo en los conceptos y reglas que rigen el negocio bancario. Este enfoque nos permitirá crear un sistema robusto, alineado con las necesidades reales del banco. Adicionalmente, aplicaremos los principios de diseño SOLID para garantizar que nuestro código sea modular, fácil de mantener y extender.

Otro aspecto clave del proyecto será la implementación de excepciones específicas orientadas al dominio, lo que nos permitirá gestionar posibles errores de manera clara y efectiva. Estas excepciones no solo ayudarán a mantener la integridad del sistema, sino que también facilitarán la identificación de problemas, mejorando la experiencia tanto para los desarrolladores como para los usuarios finales.

En resumen, este proyecto combinará un diseño centrado en el negocio con buenas prácticas de programación y un uso eficiente de tecnologías modernas, con el fin de crear una solución sólida, flexible y confiable para gestionar los servicios bancarios.

2- Requisitos.

Requisitos Funcionales (RF)

1. **Gestión de Usuarios**
   1.1. Registro de usuarios
   1.2. Crear usuario
   1.3. Buscar usuario (consulta general y específica)
   1.4. Actualizar usuario
   1.5. Borrar usuario
2. **Gestión de Clientes**
   2.1. Crear cliente
   2.2. Buscar cliente (consulta general y específica)
   2.3. Modificar cliente
   2.4. Eliminar cliente
3. **Gestión de Administradores**
   3.1. Crear administrador
   3.2. Buscar administrador (consulta general y específica)
   3.3. Modificar administrador
   3.4. Eliminar administrador
4. **Gestión de Productos**
   4.1. Gestión de Cuentas
   4.1.1. Crear cuenta
   4.1.2. Buscar cuenta
   4.1.3. Mostrar cuentas
   4.1.4. Modificar cuenta
   4.1.5. Eliminar cuenta
   4.2. Gestión de Tarjetas
   4.2.1. Crear tarjeta
   4.2.2. Buscar tarjeta
   4.2.3. Mostrar tarjetas
   4.2.4. Modificar tarjeta
   4.2.5. Eliminar tarjeta
5. **Realización de Operaciones**
   5.1. Registro de Movimientos
   5.2. Transferencias (incluye revocar transferencias)
   5.3. Pagos de tarjetas
   5.4. Ingreso de nóminas
   5.5. Domiciliación de recibos
   5.6. Realizar cambio de divisas
6. **Exportación de Datos**
   6.1. Exportar listado de clientes en formato JSON

6.2. Exportar lista de movimientos (por cliente y general) en formato JSON y PDF
6.3. Exportar lista de productos en formato JSON
6.4. Realizar copia de seguridad comprimida en formato ZIP

7. **Notificaciones en Tiempo Real**
   7.1. Notificaciones de movimientos
   7.2. Notificaciones de cambios en productos
8. **Localización y Almacenamiento**
   8.1. Importar productos desde un archivo CSV
   8.2. Almacenamiento de archivos
   8.3. Sistema de caché

Requisitos No Funcionales (RNF)

1. **Formatos de Exportación**
   - Listados de clientes, usuarios, administradores y productos deben exportarse en formato JSON.
   - Listas de movimientos deben exportarse en formato JSON y PDF.
   - Copia de seguridad comprimida en formato ZIP.
2. **Restricciones de Acceso**
   - Un cliente no puede acceder al modo administrador.
   - Un administrador no puede acceder al modo cliente.
3. **Validación y Control**
   - No se podrán realizar operaciones de salida de dinero sin saldo suficiente.
   - Los pagos no podrán superar los límites asignados.
4. **Autenticación y Autorización**
   - El sistema debe implementar autenticación y autorización usando JWT.
5. **Gestión de Movimientos**
   - La gestión de movimientos debe realizarse usando MongoDB.
6. **Documentación y Arquitectura**
   - La documentación debe realizarse con Swagger.
   - Todo el sistema debe estar desplegado en contenedores Docker.
   - Implementar excepciones adaptadas al dominio.
   - La arquitectura debe ser orientada al dominio (DDD).
7. **Pruebas y Calidad**
   - El sistema debe estar completamente testeado, incluyendo casos correctos e incorrectos.
8. **Caché**
   - Implementar un sistema de caché utilizando Redis.

Requisitos de Información (RI)

1. **Datos del Cliente**
    - Id, Nombre Completo, Dirección, Email, Teléfono, DNI, Username, Contraseña, Foto, Foto DNI, Created At, Updated At, Is Delete.
2. **Datos del Tipo de Cuenta**
    - Nombre, Interés (ej.: 2% para cuentas normales y 1% para cuentas de ahorro).
3. **Datos de la Cuenta**
    - Id, IBAN, Saldo, Fecha Apertura, Nombre, Interés, Created At, Updated At, Is Delete.
4. **Datos del Tipo de Tarjeta**
    - Tipo (crédito o débito).
5. **Datos de la Tarjeta**
    - Id, Número, Fecha Caducidad, CVV, PIN, Gasto Diario, Gasto Semanal, Gasto Mensual, Tipo, Created At, Updated At, Is Delete.
6. **Datos de los Movimientos**
    - Id Cliente, Operación (compuesta de):
        - Fecha, Importe, Destino, Origen (dependiendo si es entrada o salida).
7. **Datos del Administrador**
    - Id, Username, Contraseña, Created At, Updated At, Is Deleted.
8. **Relaciones entre Entidades**
    - Un cliente puede tener una cuenta, que puede ser normal o de ahorro, cada una con su respectivo interés.
    - Una cuenta está asociada a una tarjeta, que puede ser de crédito o débito.
    - Los movimientos se componen de operaciones, y cada operación puede ser de entrada o salida de dinero.
    - Un administrador puede modificar el interés de las cuentas.

3- Diagrama.

4- Tecnologías Usadas

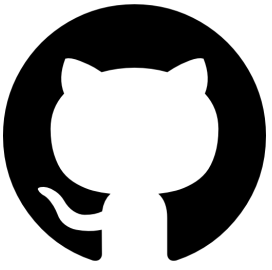
.NET

Hemos usado .NET como framework principal para el desarrollo de la aplicación	

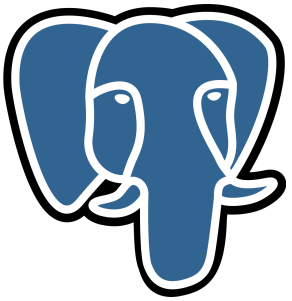
GIT

Hemos usado Git para el control de versiones del código, empleando Git Flow para llevar a cabo un desarrollo limpio y organizado.	

GITHUB

Hemos usado GitHub para albergar el repositorio de forma remota y trabajar mediante Pull Request para cada feature del programa.	

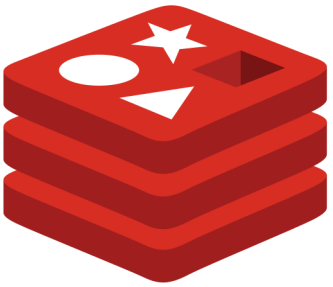
PostgreSQL

Hemos usado PostgreSQL como como base de datos para almacenar Accounts, Users, Clients, Products, Cards	

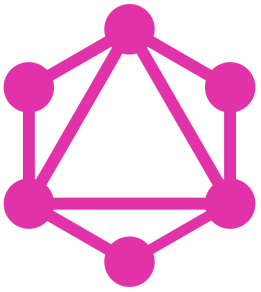
MongoDB

Hemos usado MongoDB como como base de datos para almacenar Movimientos y Domiciliaciones	

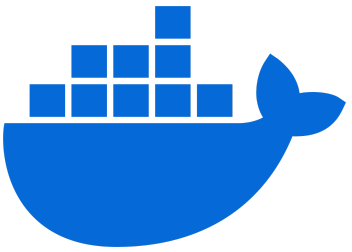
REDIS

Hemos utilizado Redis como sistema de almacenamiento en caché para mejorar el rendimiento de la aplicación	

GraphQL

Hemos utilizado GraphQL como lenguaje de consulta para nuestras APIs, permitiendo obtener exactamente los datos necesarios de manera eficiente de movimientos	

Docker

Hemos utilizado Docker para la creación y gestión de contenedores, permitiendo desplegar la aplicación	

5- Explicación del proyecto.

1-Productos.

En nuestro Banco servimos varios productos y servicios al público. Entre nuestros productos destacamos las cuentas de banco y las tarjetas de crédito. Tenemos cuentas standard, de ahorro, para empresas, inversiones etc…

En cuanto a tarjetas, tenemos de crédito, de débito, para jóvenes (mayores de 16 y menores de 18), para viajes etc…

Nuestros productos son visibles en nuestro catálogo para los usuarios autenticados en el sistema. Los usuarios podrán visualizar todos los productos o buscar por alguno en concreto.

El administrador será el único que podrá añadir nuevos productos, actualizar productos, borrarlos e importar nuevos productos en csv y exportarlos en json.

El endpoint de productos está implementado con API REST con un sistema de diferentes políticas de autorización tanto par usuarios como para administradores.

Los productos son almacenados en una base de datos PostgreSql y estos contienen:
Id(autogenerado), Nombre, Tipo Producto(Cuenta, tarjeta), CreatedAt, Updated, IsDeleted.

2-Cuentas.

En nuestra aplicación de banco, las cuentas bancarias son el núcleo de nuestras operaciones financieras. Los usuarios pueden realizar diversas acciones a través de sus cuentas, como consultar saldos, realizar transferencias, gestionar pagos, y mucho más.

Para contratar una cuenta con nuestro banco el usuario previamente registrado deberá pasar a ser un cliente en nuestra plataforma facilitando sus datos personales. Una vez forme parte del apartado de clientes podrá elegir cualquiera de los productos del tipo cuenta que se adecue a sus preferencias. Una vez contratada la cuenta el cliente podrá obtener una tarjeta y asignarla a dicha cuenta para realizar pagos con tarjeta. Una cuenta solo podrá tener asignada una tarjeta, y un cliente podrá tener varias cuentas.

Para las operaciones relacionadas con el dominio de cuentas hemos usado API REST con autorización que usa varias políticas de autorización en base al rol del usuario registrado en el sistema. Por ejemplo, un admin podrá realizar métodos GET de cuentas para obtener todas la cuentas del sistema, por id, sin mostrar datos sensibles de la cuenta del cliente. También podrá dar de bajas cuentas.

Por otro lado, el cliente podrá revisar sus cuentas, obtener una nueva cuenta y dar de baja una cuenta.

Las cuentas se almacenan en una base de datos PostgreSql y estas contienen:

Id(autogenerado), idCliente, IdProducto(producto al que referencia), idTarjeta(puede ser null), IBAN, Balance, Type(tipo de cuenta, Normal o ahorro), TAE, CreatedAt, UpdatedAt e IsDeleted.

3-Tarjetas.

Las Tarjetas de crédito de nuestro banco nos permiten realizar movimientos como pagos con tarjeta a través de la cuenta a la que estén asignadas.

Como hemos dicho anteriormente una tarjeta está asociada a una cuenta. A la hora de que un cliente quiera contratar una tarjeta, deberá de indicar el IBAN de la cuenta a la que la quiere asignar y elegir el Pin de la tarjeta, el cual debe ser un número de cuatro dígitos.

Para realizar las operaciones relacionadas con el dominio de tarjetas hemos usado una API REST con autorización que implementa diferentes políticas de autorización, tanto para cliente como para administrador.

Un administrador podrá visualizar todas las tarjetas de un banco y consultarlas por su id sin visualizar datos sensibles como el cvc o pin. También podrá exportar todas las tarjetas en json para realizar una copia de seguridad.

Un cliente podrá visualizar sus tarjetas, actualizar el pin a una tarjeta que le pertenezca dado su número de tarjeta, podrá contratar una nueva tarjeta y podrá dar de baja una tarjeta.

Las tarjetas se almacenan en una base de datos PostgreSql y de ellas almacenamos:
Id(autogenerado), número de tarjeta, idCuenta, pin, cvc, fecha de caducidad, CreatedAt, UpdatedAt e IsDeleted

4- Movimientos

Los movimientos de nuestra aplicación permiten al cliente realizar domiciliaciones, ingresos de nómina, pagos con tarjeta, transferencias y la revocación de transferencias, además de contar con diversas funciones para consultar tus movimientos.

Para las operaciones que modifican el estado (post, put, patch, delete) utilizamos una API REST con autorización, lo que implica que el usuario debe ser cliente y pasar diversas validaciones, como disponer de saldo suficiente, tener una cuenta o tarjeta activa, y realizar la operación desde una de sus cuentas. Esto garantiza que se cumplan las reglas de negocio y se eviten operaciones indebidas.

Las domiciliaciones tienen un comportamiento especial: en lugar de ejecutar el movimiento de forma inmediata, se configuran para realizar cargos periódicos según la configuración establecida por el usuario. Se incluye un mecanismo de scheduling que gestiona la ejecución recurrente de estos pagos, asegurando que se apliquen en los intervalos definidos.

Por otro lado, las consultas (operaciones GET) se implementan con GraphQL, lo que nos permite una mayor versatilidad a la hora de hacer peticiones. Con GraphQL podemos solicitar únicamente los campos necesarios, reduciendo la carga del servidor y optimizando el rendimiento, especialmente en escenarios con grandes volúmenes de datos o múltiples solicitudes concurrentes.

En resumen, hemos combinado lo mejor de ambos mundos: la robustez y seguridad de una API REST para operaciones críticas y modificatorias, y la flexibilidad y eficiencia de GraphQL para las consultas. Para el almacenamiento de todos los movimientos y domiciliaciones utilizamos MongoDB, lo que nos permite manejar de forma escalable y eficiente la información.

5- Notificaciones

Las notificaciones de nuestra aplicación son sencillas pero efectivas. Para ello, hemos diseñado un modelo general de notificación que se adapta a cada implementación específica, permitiéndonos definir el tipo de notificación y personalizar sus campos. Los campos principales de cada notificación son: el tipo (por ejemplo, Create, Update, Delete, Execute), el mensaje y la fecha de creación.

Para la implementación, utilizamos Websockets, lo que nos permite establecer una comunicación bidireccional en tiempo real entre el servidor y el cliente. Además, hemos desarrollado un diccionario atómico para gestionar la concurrencia, lo que nos permite asociar de manera segura a cada usuario autenticado con el Websocket al que se han conectado. Esto facilita la gestión de notificaciones personalizadas, garantizando que cada usuario reciba únicamente la información relevante para su sesión.

6- Usuarios

En nuestro banco, existen diferentes tipos de usuarios con distintos niveles de acceso y permisos, lo que nos permite gestionar de manera segura y eficiente las operaciones dentro del banco.

Los usuarios deben registrarse en el sistema dando sus datos personales, tras lo cual podrán autenticarse para acceder a sus funcionalidades. Dependiendo del tipo de rol que tenga el usuario.

Para la autenticación y seguridad basada en JWT, lo que garantiza sesiones seguras y permite una gestión eficiente de permisos.

Los usuarios se almacenan en una base de datos PostgreSQL y contienen los siguientes datos:

Id (autogenerado),Dni(único)Contraseña(encriptada),Rol(User, Client, Admin, Revoked), Fecha de Creación, Fecha de Actualización, Estado (Activo/Inactivo)

7- Currency

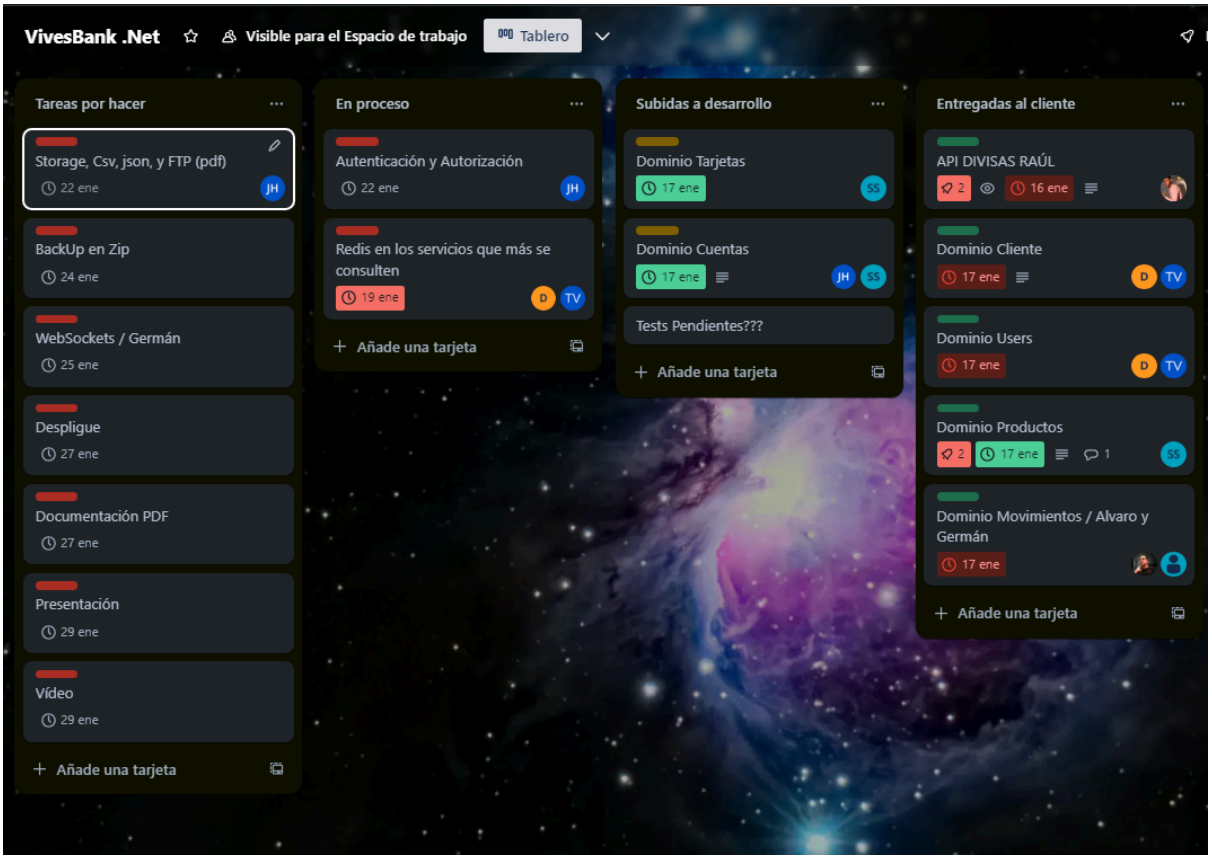
Este servicio permite la conversión de monedas en tiempo real utilizando la API de **Frankfurter**. Los usuarios y no usuarios pueden convertir un monto de una moneda a otra, obteniendo el valor actualizado basado en las tasas de cambio oficiales.

Para realizar esta conversión, el sistema se comunica con la API de **Frankfurter**, enviando solicitudes y procesando las respuestas de manera eficiente. En caso de errores, se manejan excepciones personalizadas para garantizar una experiencia fluida.

El servicio está diseñado para ser escalable y fácilmente integrable con otros sistemas. Aunque actualmente no almacena datos en una base de datos, puede adaptarse para registrar conversiones si se requiere en el futuro. Además, se pueden implementar mecanismos de autenticación para restringir el acceso a ciertas funcionalidades.

Este enfoque modular permite mantener un código limpio y organizado, facilitando la incorporación de nuevas características y asegurando un alto rendimiento en la conversión de divisas.

6- Organización Trello.



División del trabajo.

Nuestro enfoque para distribuir las tareas se ha basado en un equilibrio entre trabajos individuales y en equipo. Esto nos ha permitido optimizar el tiempo y aprovechar las fortalezas de cada miembro del grupo.

1- Trabajos en dúo:
Hemos decidido asignar grupos de dos personas. Esto nos ha permitido adelantar el tiempo, la generación de ideas conjuntas. Además, trabajar en parejas nos ha ayudado a asegurar la calidad del trabajo, ya que cada tarea pasa por dos perspectivas antes de completarse.

2- Tareas individuales:
También hemos identificado tareas específicas que se pueden realizar de forma independiente. Respetando las fortalezas individuales de cada uno y sabiendo trabajos antiguos de cada persona, asignamos a cada uno una tarea en específica que ya hubiese trabajado con ella antes.

Uso del Trello.

Gracias a Trello todo nuestro trabajo ha sido más fácil de organizar, dividimos nuestro tablero en cuatro apartados;

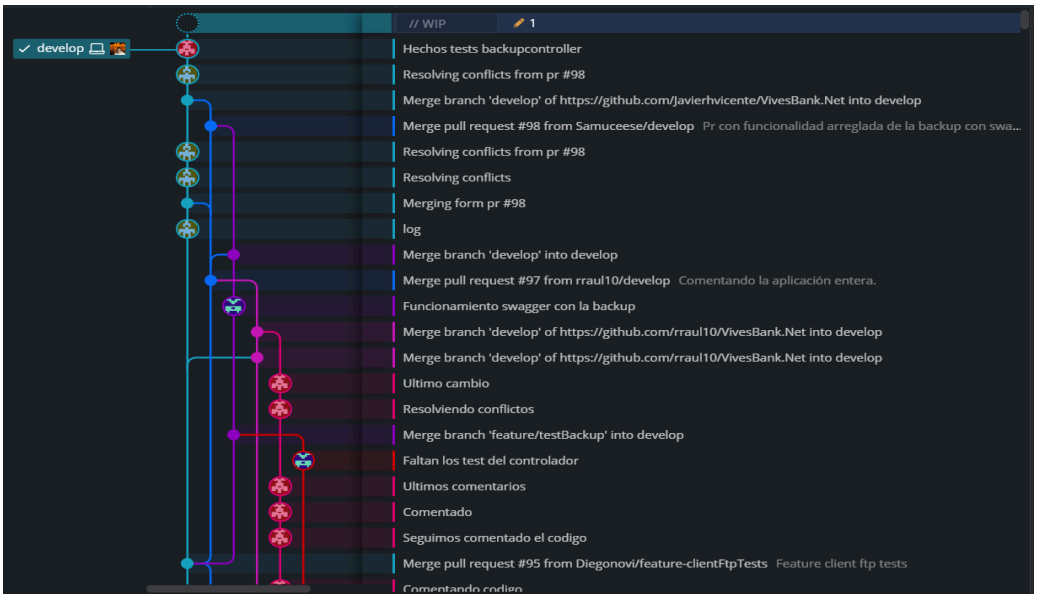
1- Tareas por hacer: En este apartado hemos dividido las tareas de cada trabajador con su nombre asignado al lado de la tarea.

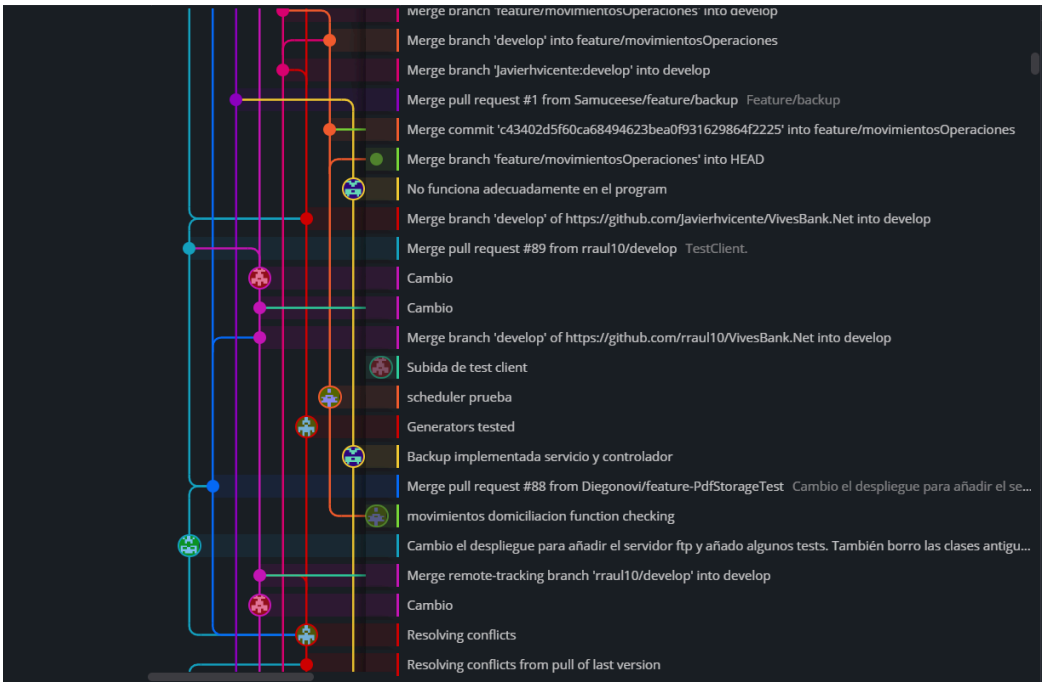
2- En proceso: Aquí íbamos poniendo cada tarea asignada a su trabajador en el momento que este estuviese haciéndola.

3- Subidas a desarrollo: Una vez terminadas las tareas, comprobaremos que se hubiesen añadido bien al programa y funcionasen correctamente.

4- Entregadas al cliente: Ya habiendo comprobado todo el funcionamiento, las subíamos a este apartado para entregarla al cliente.

7- Git Flow.





GitFlow es un flujo de ramas en Git diseñada para manejar el desarrollo de software de manera organizada. En la imagen, se observa un historial de commits con múltiples ramas y fusiones (merge). Gracias a ello, podemos trabajar cada uno paralelamente a los otros compañeros sin pisar los cambios.

8- Costes

ESTIMACIÓN DE COSTES DE LA APLICACIÓN

Resumen Costes Infraestructura (mensuales)	
Licencias software Empresa (IDEs, aplicaciones auxiliares, sistemas operativos)	400,00 €
Licencia IDE jetBrains All products pack para organizaciones	77,90 €
Amortizaciones ordenadores portátiles (7 portátiles ASUS VivoBook)	51,43 €
Margen para costes imprevistos (10% del total de costes iniciales)	40,00 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	85,40 €
Total Costes Infraestructura mensuales	**654,73 €**

Resumen Costes Propios de la aplicación (mensuales)	
Alojamiento del proyecto en servidor remoto	250,00 €
Medidas seguridad de la aplicación	150,00 €
Mantenimiento post entrega	100,00 €
Atención al cliente	50,00 €
Margen para costes imprevistos (10% del total de costes iniciales)	55,00 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	90,75 €
Total Costes aplicación mensuales	**695,75 €**

Costes Iniciales de la aplicación	
Total coste tareas RRHH	38.625,00 €
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 (10 horas)	3.000,00 €
Manuales de uso	500,00 €
Margen para costes imprevistos (10% del total de costes iniciales)	4.487,50 €
Margen de Beneficio (15% del total inicial incluyendo imprevistos)	7.929,38 €
Total Costes iniciales aplicación	**60.791,88 €**
Coste hora Formador	300,00 €
Incluye Salario + Coste de Empresa	

Estimación de costes Aplicación	
1. Costes iniciales de la aplicación	**60.791,88 €**
2. Costes mensuales de la aplicación	**1.350,48 €**

COSTES RRHH

Horas asignadas por Requisitos Funcionales		Horas Estimada	Coste Hora	Total €
	Análisis y Diseño proyecto	20	125,00 €	2.500,00 €
	Despliegue de la aplicación	15	125,00 €	1.875,00 €
	Despliegue de informes de tests	5	125,00 €	625,00 €
	Creación de la documentación del proyecto	10	125,00 €	1.250,00 €

Horas asignadas por Requisitos Funcionales

Concepto/RF	Nombre	Horas Estimadas
RF1	Gestión Usuarios	
RF1.1	Registro de Usuarios	6
RF1.2	Crear usuario	7
RF1.3	Buscar usuario (consulta general y específica	5
RF1.4	Actualizar usuario	6
RF1.5	Borrar usuario	5
RF2	Gestión de Clientes	
RF2.1.1	Crear cliente	7
RF2.1.2	Buscar cliente	5
RF2.1.3	Modificar cliente	6
RF2.1.4	Eliminar cliente	7
RF3	Gestión de administradores	
RF 3.1	Crear administrador	4
RF 3.2	Buscar administrador (consulta general y específica)	4
RF 3.3	Modificar administrador	4
RF 3.4	Eliminar administrador	4
RF4	Gestión Productos	
RF4.1	Gestión de Cuentas	
RF4.1.1	Crear Cuenta	6
RF4.1.2	Buscar Cuenta	6
RF4.1.3	Mostrar Cuentas	6
RF4.1.4	Modificar Cuenta	6
RF4.1.5	Eliminar Cuenta	6

RF4.2	Gestión de Tarjetas	
RF4.2.1	Crear Tarjeta	5
RF4.2.2	Buscar Tarjeta	5
RF4.2.3	Mostrar Tarjetas	5
RF4.2.4	Modificar Tarjeta	5
RF4.2.5	Eliminar Tarjeta	5
RF5	Realización de Operaciones	
RF 5.1	Registro de Movimientos	7
RF 5.2	Transferencias (y revocación)	15
RF 5.3	Pagos de tarjetas	5
RF 5.4	Ingreso de nóminas	5
RF 5.5	Domiciliación de recibos	5
RF 5.6	Realizar cambio de divisas	10
RF 6	Exportación de Datos	
RF 6.1	Exportar clientes a formato JSON	5
RF 6.2	Exportar movimientos a formato JSON y PDF	10
RF 6.3	Exportar productos a formato JSON	5
RF 6.4	Realizar copia de seguridad formato ZIP	12
RF 7	Notificaciones en Tiempo Real	10
RF 7.1	Notificaciones de movimientos	
RF 7.2	Notificaciones de cambios en productos	
RF 8	Localización y Almacenamiento	
RF 8.1	Importar productos desde un archivo CSV	5
RF 8.2	Almacenamiento de archivos	20
RF 8.3	Sistema de caché	20
	Total horas	309
	Total coste tareas RRHH	38.625,00 €

AMORTIZACIONES

Equipo	Precio Compra	Período Amortización (años)	Amortización Anual	Amortización Mensual	Unidades	Amortización Total Mensual
Portátil ASUS VivoBook	529,00 €	6	88,17 €	7,35 €	7	51,43 €