



VIVES BANK

#### Raúl Fernández, Javier Hernández, Yahya el Hadri, Samuel Cortés, Javier Ruíz.

INDICE

1. [Arquitectura del proyecto.](#_bookmark0)
   * 1. [1.1- Patrón de diseño.](#_bookmark1)
2. [Objetivos generales y específicos del proyecto.](#_bookmark2)
3. [Tecnologías usadas.](#_bookmark3)
4. Uso de la aplicación
5. [Casos de uso](#_bookmark4)

# Arquitectura del proyecto.



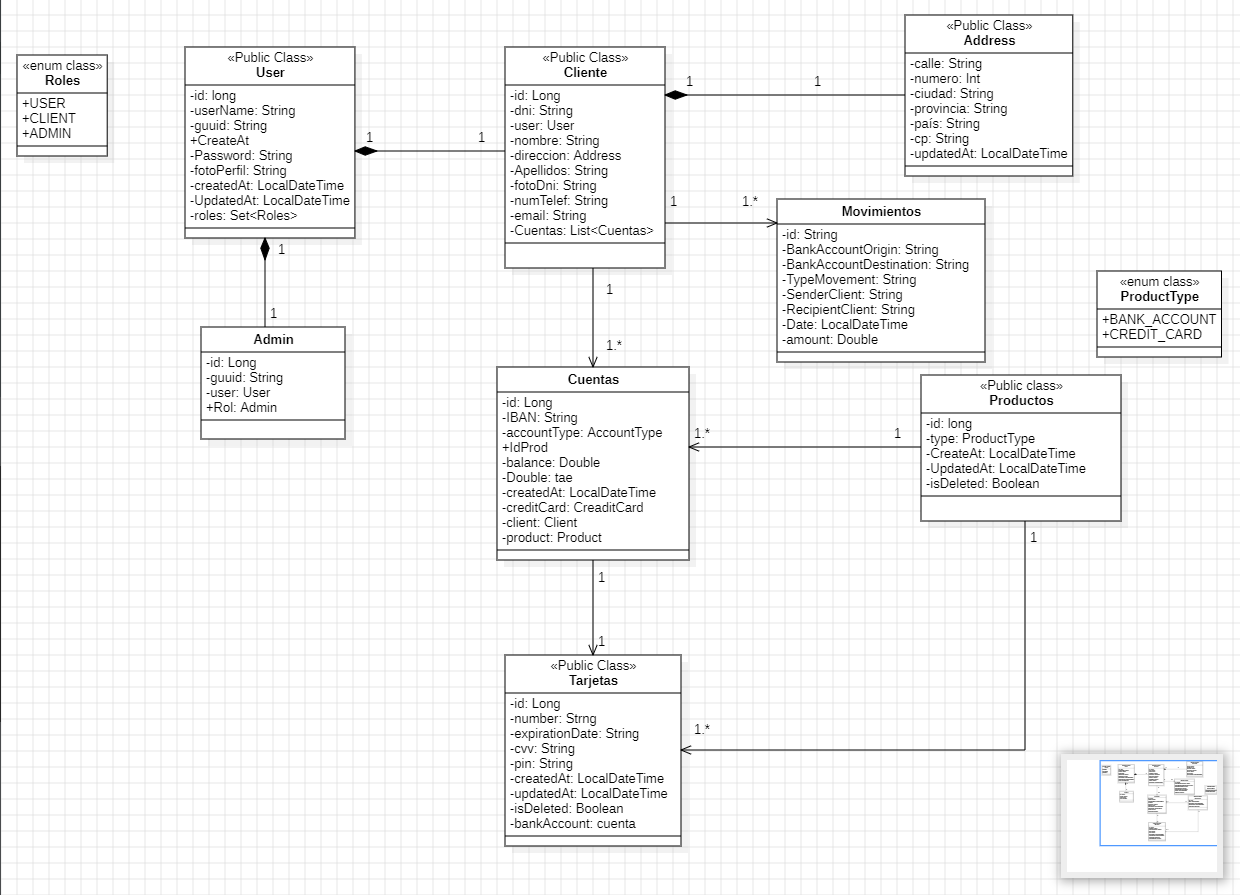
Nuestra aplicación realiza la gestión de productos, tarjetas, cuentas, movimientos usuarios, clientes y administradores de un banco, además de un servicio de cambio de divisas. Para ello hemos usado una arquitectura centrada al dominio ya que hemos pensado que era la mejor forma de organizar el código de una forma clara, con la intención de tener dividido el código en diferentes servicios.

El modelo de dominio es una representación simplificada y estructurada de la lógica de negocio. En lugar de centrarse en cómo funcionan los detalles técnicos (como bases de datos o sistemas externos), el modelo de dominio refleja directamente los conceptos y reglas del negocio, por ejemplo, clases como Usuario, Tarjeta o Cliente.

Hemos elegido la arquitectura orientada al domino ya que a la hora de desarrollar el software de la aplicación el lenguaje común ayuda a evitar malentendidos y garantiza que todos tengan una comprensión clara de lo que el software debe hacer. Además, esto permite modificar o escalar partes del sistema sin afectar el todo.

Un cliente está compuesto por un usuario. Como cliente este podrá contratar varios servicios. Un cliente deberá tener una cuenta bancaria como mínimo, ya que para ser cliente debe contratar un servicio. Una cuenta bancaria solo podrá tener una tarjeta de crédito asociada. Como cliente este podrá hacer varios movimientos, como pagos y transferencias o recibir movimientos como pagos y nóminas.

Como administrador podremos gestionar usuarios, clientes, administradores, productos, cuentas bancarias, tarjetas y movimientos.



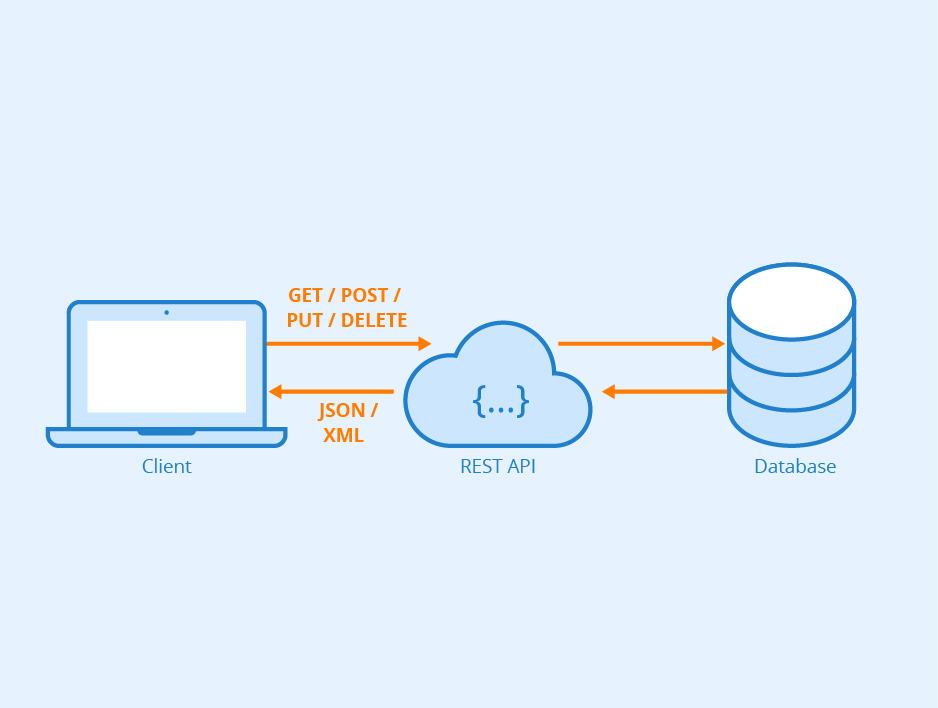
# 1.1- Patrón de diseño.

Para el desarrollo del proyecto hemos utilizado el patrón de diseño facade en el cual el cliente se comunica con la aplicación a través de una API Rest. Dicha API Rest se comunicará con los controladores del sistema que a través de los servicios realizaran las operaciones necesarias almacenando los datos en una base de datos a través de los repositorios.

Nosotros usamos una base de datos local PostgreSql para almacenar usuarios, administradores, clientes, productos, cuentas y tarjetas. Para movimientos, usamos una base de datos Mongo.

Nuestra aplicación cuenta con un sistema de seguridad que actúa como middleware entre la API y la aplicación. Cualquier persona puede consultar nuestros productos, pero para contratar cualquiera de ellos deberá darse de alta en el sistema. A su vez como cliente solo podrá consultar los productos que tenga contratados y los movimientos en los que participé. El encargado de la gestión de los usuarios, clientes y productos será el administrador del sistema.

Nuestro sistema de seguridad trabaja con spring security, JWT y Auth0 para la generación de tokens y autenticación del usuario en el sistema.



# Objetivos generales y específicos del proyecto.

El **objetivo general** de este proyecto es el de crear una aplicación java con springBoot en el cual podamos gestionar, usuarios, clientes y administradores. También, podremos gestionar productos, cuentas bancarias, tarjetas de crédito y movimientos. Añadiendo un servicio de cambio de divisas tanto para los clientes y usuarios del banco como para cualquier persona que quiera usarlo aunque no se haya dado de alta en el sistema.

Requisitos funcionales:

1.- Gestion de usuarios

1.1. – Registro de usuarios

1.2.- Creación

1.3.- Consulta, general y específica

1.4.- Modificación

1.5.- Borrado

2.- Gestión de clientes

2.1.- Creación

2.2.- Consulta, general y específica

2.3.- Modificación

2.4.- Borrado

3.- Gestión de admins

3.1.- Creación

3.2.- Consulta, general y específica

3.3.- Modificación

3.4.- Borrado

4.- Gestión de productos

4.1.- Creación

4.2.- Consulta, general y específica

4.3.- Modificación

4.4.- Borrado

5.- Realización de operaciones

5.1.- Transferencias

5.2.- Pagos

5.3.- Domiciliación de recibos

5.4.- Ingreso de nómina

6.- Exportación de datos

6.1.- Exportación de listado de clientes

6.2.- Exportación de lista de movimientos, por cliente y general

6.3.- Copia de seguridad

7.- Notificaciones de movimientos

8.- Notificaciones de cambios en productos

9.- Cambio de divisas

10.- Localización de cajeros

11.- Importación de productos

12.- Validación de datos de entrada

13.- Almacenamiento de archivos

14.- Sistema de caché

15.- Autenticación

16.- Autorización

Requisitos no funcionales:

Exportación de listado de clientes en formato JSON.

Exportación de listado de usuarios en formato JSON.

Exportación de listado de administradores en formato JSON.

Exportación de lista de movimientos en JSON y PDF.

Exportación de lista de productos en JSON.

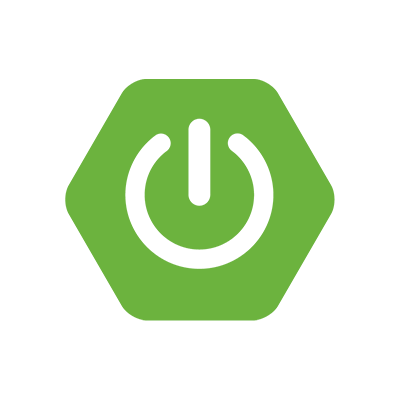
Copia de seguridad comprimida en ZIP.

Importación de productos desde un CSV.

Caché en REDIS.

Autenticacion y autorizacion con JWT.

# Tecnologías usadas.

**Spring Boot**

Hemos usado Spring Boot con java como framework principal para el desarrollo de la aplicación.

**DESCRIPCIÓN**

# 

**Git**



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

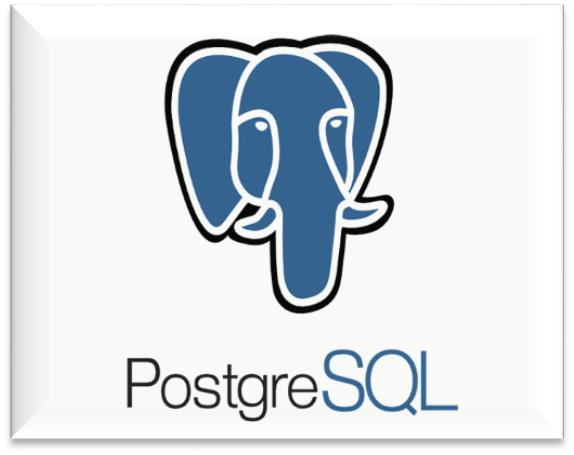
**DESCRIPCIÓN**

**GitHub**

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

**DESCRIPCIÓN**

**PostgreSQL**



Hemos usado PostgreSql como base de datos de producción para el almacenamiento de usuarios, clientes, administradores, productos, tarjetas y cuentas bancarias.

**DESCRIPCIÓN**

**Mongo**

Hemos usado Docker para el despliegue en contenedores de la aplicación y para el almacenamiento de la base de datos Mongo. Docker no ha ayudado a ayudado a realizar el despliegue con un proxy inverso para mostrar la documentación del proyecto.

**DESCRIPCIÓN**

**Docker**



Hemos usado Docker para el despliegue en contenedores de la aplicación y para el almacenamiento de la base de datos Mongo. Docker no ha ayudado a ayudado a realizar el despliegue con un proxy inverso para mostrar la documentación del proyecto.

**DESCRIPCIÓN**

**Nginx**

Hemos usado nginx para servir el proxy inverso de nuestra aplicación y así poder mostrar la documentación.

**DESCRIPCIÓN**





**Redis**



Hemos usado redis como cache de nivel 2 para clientes, movimientos y productos, ya que son los datos que más se van a consultar en la aplicación y así descargamos las peticiones a la base de datos.

**DESCRIPCIÓN**

# 4. Uso de la aplicación

Endpoints del proyecto:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acción | Petición | URL | Cod Respuesta | ROL |
| SignUp | POST | /vivesbank/v1/auth/signip | 201, 400 | No es necesario |
| SignIn | POST | /vivesbank/v1/auth/signin | 201,400 | No es necesario |
| Todos users | GET | /vivesbank/v1/users | 200 / 401/403 / 404 | admin |
| User by id | GET | vivesbank/v1/users/{id} | 200 / 404/ 401 | admin |
| Get me profile as user | GET | vivesbank/v1/users/me/profile | 200 / 404/ 401/403 | User, admin |
| Get by userName | GET | vivesbank/v1/users/name | 200 / 404/ 401 | admin |
| Create user | POST | vivesbank/v1/users | 201 / 400 / 401 / 40 | admin |
| Update user | PUT | vivesbank/v1/users/{id} | 201/404 / 400 / 403/ 401 | admin |
| Update me profile | PUT | vivesbank/v1/users/me/profile | 201/ /404 / 400 /403/401 | User, admin |
| Borrar Usuario | DELETE | vivesbank/v1/users/{id} | 204 /404 /401/403 | admin |
| Borrar me profile | DELET | vivesbank/v1/users/me/profile | 204/404/401/403 | user, admin |
| Todos Admin | GET | vivesbank/v1/admins | 200 / 401 / 403/ 404 | admin |
| Admin by Id | GET | vivesbank/v1/admins/{id} | 200 / 401 / 403 / 404 | admin |
| Create admin | POST | vivesbank/v1/admins | 200 / 401 / 403 / 400 | admin |
| Update admin | PUT | vivesbank/v1/admins/{id} | 200 / 401 / 403 / 400 / 404 | admin |
| Delete admin | DELETE | vivesbank/v1/admins/{id} | 200 / 401 / 403 / 400 / 404 | admin |
| Todos clientes | GET | vivesbank/v1/clients | 200 / 404 / 401 / 403 | admin |
| GetBy dni | GET | vivesbank/v1/clients/dni/{dni} | 200 / 404 / 401 / 403 | admin |
| Cliente by id | GET | vivesbank/v1/clients/{id} | 200 / 404 / 401 / 403 | admin |
| Get my profile as cliente | GET | vivesbank/v1/clients/me/profile | 200 / 404 / 401 / 403 | client |
| Create cliente | POST | vivesbank/v1/clients | 201 / 401 / 400/ 403 | user |
| Update cliente | PUT | vivesbank/v1/clients/{id} | 200 /404 / 401 | admin |
| Update my profile | PUT | vivesbank/v1/clients/me/profile | 201/404 / 401 / 403 | cliente |
| Update my dni | PATCH | vivesbank/v1/clients/me/profile/dni | 201/404 / 401 / 403 | cliente |
| Update my photo | PATCH | vives/v1/clients/me/profile/perfil |  |  |
| Borrar Cliente por id | DELETE | vivesbank/v1/clients/{id} | 204 /404 / 401 / 403 | admin |
| Borrar mi perfil como cliente | DELETE | vivesbank/v1/clients/me/profile | 204 / 404/ 401 / 403 | admin |
| GetAll productos | GET | vivesbank/v1/products | 200 / 404 |  |
| GetById | GET | vivesbank/v1/products/{id} | 200/ 404 |  |
| GetByCategoria | GET | vivesbank/v1/products/type /{type} | 200 / 404 |  |
| CreateProducto | POST | vivesbank/v1/products | 200 / 404 / 401 | admin |
| UpdateProducto | PUT | vivesbank/v1/productos/1 | 200 / 404 / 401 | admin |
| deleteProducto | DELETE | vivesbank/v1/productos/1 | 200 / 404 / 401 | admin |
| GetAll | GET | vivesbank/v1/accounts | 200 / 404 / 401 / 403 | admin |
| Get by id | GET | vivesbank/v1/accounts/{id} | 200 / 404 / 401 / 403 | admin |
| Get all acounts by UserId | GET | vivesbank/v1/accounts/ allAccounts/{id} | 200 / 404 / 401 / 403 | admin |
| Get my cuentas | GET | vivesbank/v1/accounts/me | 200 / 404 / 401 / 403 | cliente |
| CreateCuenta | POST | vivesbank/v1/accounts | 200 / 404 / 401 | cliente |
| Delete cuenta by id | DELETE | vivesbank/v1/accounts/{id} | 200 / 404/ 401 / 403 | admin |
| Delete me Cuenta | DELETE | vivesbank/v1/accounts/me/{id} | 200 / 404 / 401 / 403 | cliente |
| GetAllTarjetas | GET | vivesbank/v1/creditcard | 200 / 404 / 401/403 | admin |
| Get me tarjetas | GET | Vivesbank/v1/creditcard/me/creditcard | 200/404/401/403 | cliente |
| GetAllTarjetasByUserID | GET | /vivesbank/v1/creditcard/id/{id} | 200 / 404 / 401/403 | admin |
| GetCredit card by date | GET | /vivesbank/v1/creditcard/date/{date }  /vivesbank/v1/creditcard/date/before/{date} | 200 / 404 / 401/403 | admin |
| CreateTarjeta | POST | /vivesbank/v1/creditcard | 200/400 /401/403 | cliente |
| Delete my tarjeta | DELETE | /vivesbank/v1/creditcard/me/creditcard | 204 / 404/ 401/403 | cliente |
| DeleteTarjeta | DELETE | /vivesbank/v1/creditcard/{id} | 204 /404/ 401/403 | cliente |
| Create movement | POST | vivesbank/v1/movements | 201 / 404 / 401 / 403 | CLIENT |
| Delete movement | DELETE | vivesbank/v1/movements/{id} | 204 / 404 / 401 / 403 | admin |
| Ver movimientos de un cliente | GET | vivesbank/v1/movements/admin/client/{clientId} | 200 / 404 / 401 / 403 | admin |
| Ver movimientos de mi cuenta | GET | vivesbank/v1/me/movements/ | 200 / 404 / 401 / 403 | CLIENT |
| Ver movimiento por ID (admin) | GET | vivesbank/v1/movements/admin/{movementId} | 200 / 404 / 401 / 403 | ADMIN |
| Ver movimiento específico (cliente) | GET | vivesbank/v1/me/{clientId}/{movementId} | 200 / 404 / 401 / 403 | CLIENT |
| Ver movimientos recibidos (admin) | GET | |  | | --- | | vivesbank/v1/movements/admin/RecipientMovements/{clientId} |  |  | | --- | |  | |  | | 200 / 404 / 401 / 403 | admin |
| Ver movimientos recibidos (cliente) | GET | vivesbank/v1/me/RecipientMovements | 200 / 404 / 401 / 403 | CLIENT |
| Ver movimientos enviados (admin) | GET | vivesbank/v1/movements/admin/SentMovements/{clientId} | 200 / 404 / 401 / 403 | ADMIN |
| Ver movimientos enviados (cliente) | GET | vivesbank/v1/me/SentMovements | 200 / 404 / 401 / 403 | CLIENT |
| Ver movimientos por tipo (admin) | GET | vivesbank/v1/movements/admin/type/{clientId}/{typeMovement} | 200 / 404 / 401 / 403 | ADMIN |
| Ver movimientos por tipo (cliente) | GET | vivesbank/v1/me/movement/type/{typeMovement} | 200 / 404 / 401 / 403 | CLIENT |
| Eliminar movimiento por ID (admin) | DELETE | vivesbank/v1/movements/admin/{movementId} | 200 / 404 / 401 / 403 | ADMIN |
| Eliminar movimiento propio (cliente) | DELETE | vivesbank/v1/me/movement/{movementId} | 200 / 404 / 401 / 403 | CLIENT |
| Generar PDF de movimientos (admin) | GET | vivesbank/v1/movements/pdf | 200 / 404 / 401 / 403 | ADMIN |
| Generar PDF por movimiento (admin) | GET | vivesbank/v1/movements/pdf/{id} | 200 / 404 / 401 / 403 | ADMIN |
| |  | | --- | | Generar PDF de mis movimientos |  |  | | --- | |  | | GET | vivesbank/v1/me/pdf | 200 / 404 / 401 / 403 | CLIENT |
| PDF de movimientos enviados (cliente) | GET | vivesbank/v1/me/pdf/sended | 200 / 404 / 401 / 403 | CLIENT |
| PDF de movimientos recibidos (cliente) | GET | vivesbank/v1/me/pdf/recibed | 200 / 404 / 401 / 403 | CLIENT |
| |  | | --- | | Listar todas las monedas | | |  | | --- | | GET | | |  | | --- | | /currency | | |  | | --- | | 200 / 404 / 401 | | |  | | --- | |  | |
| Detalles de moneda por símbolo | GET | /currency/{symbol} | 200 / 404 / 401 / 403 |  |
| Historial de moneda | GET | /currency/history/{date} | 200 / 404 / 401 / 403 |  |
| Conversión de moneda | GET | /currency/convert | 200 / 404 / 401 / 403 |  |
| Series temporales | GET | /currency/timeseries | 200 / 404 / 401 / 403 |  |
| Últimos tipos de cambio | GET | /currency/latest | |  | | --- | | 200 / 404 / 401 |  |  | | --- | |  | |  |
| Listar monedas disponibles | GET | /currency/currencies | 200 / 404 / 401 |  |

# CASOS DE USO.

Nombre: Darse de alta en el sistema

Id: CU 1



Requisito funcional: Se podrá registrar usuarios en el sistema mediante un servicio de autenticación con JWT

Precondición:

Postcondición: El sistema debe haber registrado al nuevo usuario y haber devuelto su token de sesión.

Descripción: Se puede registrar como usuario mediante POST /vivesbank/v1/auth/singup , introduciendo los datos necesarios para la creación del nuevo usuario. Username (email), contraseña (8 caracteres mínimo, mayúscula, minúscula, número y carácter especial), password check, foto de perfil.

Secuencia normal:

1. El usuario se registra introduciendo todos los datos correctamente

2. El sistema procesa la petición

3. El sistema devuelve el token de autenticación.

Secuencia alternativa:

2.1 El cliente introduce datos incorrectos para registrarse como puede ser un email no válido. Lanza AuthSignUpInvalid exception, código 400.

2.2 El cliente introduce datos incorrectos para registrarse como puede ser una contraseña no válida. Lanza UserPasswordBadRequest exception código 400.

2.3 El cliente introduce datos incorrectos para registrarse como puede ser que las contraseñas no coincidan. Lanza UserDiferentePasswords exception código 400.

2.4 El sistema no puede procesar la petición ya que ya existe un cliente con el mismo username(email). Lanza UserAuthNameOrEmailExisten exception, código 400.

Actores: Usuario no registrado

Nombre: Iniciar sesión en el sistema

Id: CU 1.1

Requisito funcional: Se podrá iniciar sesión como un usuario previamente registrado en la base de datos.

Precondición: Tener un perfil de usuario registrado en el sistema



Postcondición: El sistema debe haber encontrado al usuario que inicia sesión y le dará el token de sesión

Descripción: Se puede iniciar sesión como usuario mediante POST /vivesbank/v1/auth/sinIn , introduciendo los datos necesarios para el inicio de sesión, username(email) y la contraseña. Una vez registrados tendremos nuestro token de sesión y podremos realizar las operaciones permitidas para el usuario.

Secuencia normal:

1. El usuario inicia sesión introduciendo todos los datos correctamente

2. El sistema procesa la petición

3. El sistema devuelve el token de autenticación.

Secuencia alternativa:

2.1 El usuario introducido no existe en el sistema por lo que se lanza AuthSignUpInvalid exception y código de error 404.

2.2 El usuario se ha encontrado per no coinciden las contraseñas, por lo tanto no se puede autenticar al usuario que inicia sesión. Se lanza BadCredentialsException y código de error 400.

Actores: Usuario no registrado



Nombre: Consultar mis datos de usuario

Id: CU 2

Requisito funcional: Como usuario podré consultar mis datos personales.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema debe haber encontrado al usuario y debe mostrar sus datos en la respuesta de la petición.

Descripción: Se puede consultar los datos del usuario que esta activo y registrado en el sistema mediante GET /vivesbank/v1/users/me/profile.

Secuencia normal:

1. El usuario manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve la respuesta con los datos del usuario.

Secuencia alternativa:

2.1 El usuario introducido no existe en el sistema por lo que se lanza UserNotFound exception y código de error 404.

Actores: Usuario

Nombre: Actualizar mis datos de usuario

Id: CU 2.1

Requisito funcional: Como usuario podré actualizar mis datos personales.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema debe haber encontrado al usuario, debe haber validado los cambios y debe haber mostrado los cambios realizados al usuario en la respuesta de la petición

Descripción: Se puede consultar los datos del usuario que esta activo y registrado en el sistema mediante PUT /vivesbank/v1/users /me/profile. Podrá cambiar su username(correo), contrseña, y fotoPerfil

Secuencia normal:

1. El usuario manda la petición al sistema, añadiendo los campos que quiere cambiar en la petición.

2. El sistema procesa la petición

3. El sistema devuelve la respuesta con los datos del usuario actualizados

Secuencia alternativa:

2.1 El usuario introducido no existe en el sistema por lo que se lanza UserNotFound exception y código de error 404.

2.2 El nombre de usuario no es válido ya que no es un email válido, se devuelve código 400

2.3 La contraseña no es válida ya que no tiene 8 caracteres como mínimo, mayúscula, minúscula, número y carácter especial, se devuelve código 400

Actores: Usuario

Nombre: Darme de baja del sistema

Id: CU 2.2

Requisito funcional: Como usuario me podré dar de baja en el sistema.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema debe haber registrado al usuario como borrado y debe de haberle revocado los permisos para que no pueda volver a acceder con esas credenciales.

Descripción: Un usuario se puede dar de baja mediante DELETE /vivesbank/v1/users /me/profile, perdiendo el acceso al sistema con esos credenciales.

Secuencia normal:

1. El usuario manda la petición al sistema

2. El sistema procesa la petición, revocando el acceso al usuario.

3. El sistema devuelve la respuesta 204 indicando que se ha dado de baja.

Secuencia alternativa:

2.1 El usuario introducido no existe en el sistema por lo que se lanza UserNotFound exception y código de error 404.

Actores: Usuario

Nombre: Obtener todos los usuarios

Id: CU 2.3

Requisito funcional: Como administrador podré consultar a todos los usuarios del sistema.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe devolver a todos los usuarios almacenados en la base de datos

Descripción: Un administrador podrá consultar a todos los usuarios mediante GET/vivesbank/v1/users.

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve los usuarios registrados en el sistema.

Secuencia alternativa:

2.1 No se encuentran usuarios en la base de datos. Se lanza error 404.

Actores: Administrador

--

Nombre: Obtener un usuario por su guuid

Id: CU 2.4

Requisito funcional: Como administrador podré consultar a un usuario registrado en el sistema por su guuid.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe devolver el usuario en caso de haber sido encontrado

Descripción: Un administrador podrá consultar a cualquier usuario pasando su guuid GET/vivesbank/v1/users/{id}.

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve el usuario.

Secuencia alternativa:

2.1 No se encuentran usuario en la base de datos con ese guuid. Se lanza UserNotFound Exception y error 404

Actores: Administrador

Nombre: Obtener un usuario por su username

Id: CU 2.5

Requisito funcional: Como administrador podré consultar a un usuario registrado en el sistema por su username.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe devolver el usuario en caso de haber sido encontrado.

Descripción: Un administrador podrá consultar a cualquier usuario pasando su username(email) GET/vivesbank/v1/users/name/{name}.

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve el usuario.

Secuencia alternativa:

2.1 El sistema no puede procesar la petición ya que el username(email) no tiene un formato válido, se lanza código 400.

2.2 No se encuentran usuario en la base de datos con ese username. Se lanza UserNotFound Exception y error 404.

Actores: Administrador

Nombre: Crear un usuario

Id: CU 2.6

Requisito funcional: Como administrador podré crear a un en el sistema.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe devolver el usuario en caso de haber sido creado, código 201.

Descripción: Un administrador podrá crear a un usuario pasando su username(email), contraseña, fotoPerfil, y su estado de borrado isDeleted(false) con POST /vivesbank/v1/users .

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve el usuario y lo guarda en la base de datos.

Secuencia alternativa:

2.1 El sistema no puede procesar la petición ya que el username(email) no tiene un formato válido, se lanza código 400.

2.2 El sistema no puede procesar la petición ya que la contraseña no tiene un formato válido, se lanza código 400.

2.3 En caso de que se quiera guardar un usuario con el mismo username se lanzará código 409.

Actores: Administrador

Nombre: Actualizar a un usuario por su id.

Id: CU 2.7

Requisito funcional: Como administrador podré actualizar a un usuario del sistema.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe devolver el usuario en caso de haber sido actualizado.

Descripción: Un administrador podrá actualizar a un usuario pasando su guuid, username(email), contraseña, fotoPerfil, y su estado de borrado isDeleted(false) con PUT /vivesbank/v1/users/{id}.

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve el usuario y lo guarda en la base de datos.

Secuencia alternativa:

2.1 El sistema no puede procesar la petición ya que el username(email) no tiene un formato válido, se lanza código 400.

2.2 El sistema no puede procesar la petición ya que la contraseña no tiene un formato válido, se lanza código 400.

2.3 En caso de que se quiera guardar un usuario con el mismo username se lanzará código 409.

2.4 El sistema no encuentra a usuario con el guuid indicado, lanza UserNotfound 404.

Actores: Administrador

Nombre: Borrar a un usuario por su id.

Id: CU 2.8

Requisito funcional: Como administrador podré borrar a un usuario del sistema.

Precondición: Estar registrado en el sistema como administrador y tener un token de autenticación activo.

Postcondición: El sistema debe mostrar al usuario como borrado y debe de haber revocado los permisos para el uso del sistema al usuario con esas credenciales.

Descripción: Un administrador podrá borrar de forma lógica a un usuario pasando su guuid con Delete /vivesbank/v1/users/{id}.

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición

3. El sistema devuelve 204 indicando que la operación se ha realizado con éxito.

Secuencia alternativa:

2.1 El sistema no encuentra a usuario con el guuid indicado, lanza UserNotfound 404.

Actores: Administrador

## CU 3.1

Requisito funcional: Como administrador podré buscar las cuentas del banco.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo y rol de administrador.

Postcondición: El sistema deberá devolver una lista de todas las cuentas del banco.

Descripción: Un administrador puede obtener la lista de todas las cuentas del banco

GET /vivesbank/v1/accounts

Secuencia normal:

1. El administrador manda la petición al sistema

2. El sistema procesa la petición, permitiendo el acceso al administrador,

3. El sistema devuelve la respuesta 200, indicando que se ha devuelto la lista correctamente.

Secuencia alternativa:

2.1 No exista el usuario con rol de administrador en el sistema

Actores: Administrador

----------------------------------------------------------------------

## CU 3.2

Requisito funcional: Como administrador podré buscar una cuenta bancaria específica por su ID.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo y rol de administrador.

Postcondición: El sistema deberá devolver los detalles de la cuenta bancaria solicitada, o un mensaje de error si la cuenta no existe.

Descripción: Un administrador puede obtener la información detallada de una cuenta bancaria específica proporcionando su ID único.

GET /vivesbank/v1/accounts/{id}

Secuencia normal:

1. El administrador envía una petición al sistema incluyendo el ID de la cuenta que desea consultar.

2. El sistema verifica que el usuario tenga el rol de administrador y que el token de autenticación sea válido.

3. El sistema busca en la base de datos la cuenta con el ID proporcionado.

4. Si la cuenta es encontrada, el sistema devuelve los detalles de la cuenta en la respuesta con un código de estado 200.

5. Si la cuenta no es encontrada, el sistema devuelve un mensaje de error con un código de estado 404.

Secuencia alternativa:

2.1 El usuario no tiene el rol de administrador o el token de autenticación no es válido.

\* El sistema devuelve un mensaje de error con un código de estado 403 (Forbidden), indicando que el usuario no tiene autorización para realizar esta acción.

Actores: Administrador

------------------------------------------------------------------------

## CU 3.3

Requisito funcional: Como administrador podré buscar todas las cuentas bancarias asociadas a un cliente específico.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo y rol de administrador.

Postcondición: El sistema deberá devolver una lista de todas las cuentas bancarias asociadas al cliente especificado, o un mensaje de error si el cliente no existe.

Descripción: Un administrador puede obtener una lista de todas las cuentas bancarias asociadas a un cliente específico, proporcionando el ID del cliente.

GET /vivesbank/v1/accounts/allAccounts/{id}

Secuencia normal:

1. El administrador envía una petición al sistema incluyendo el ID del cliente cuyas cuentas desea consultar.

2. El sistema verifica que el usuario tenga el rol de administrador y que el token de autenticación sea válido.

3. El sistema busca en la base de datos al cliente con el ID proporcionado.

4. Si el cliente es encontrado, el sistema busca todas las cuentas bancarias asociadas a ese cliente.

5. El sistema devuelve una lista de las cuentas encontradas en la respuesta con un código de estado 200.

6. Si el cliente no es encontrado, el sistema devuelve un mensaje de error con un código de estado 404.

Secuencia alternativa:

2.1 El usuario no tiene el rol de administrador o el token de autenticación no es válido.

\* El sistema devuelve un mensaje de error con un código de estado 403 (Forbidden), indicando que el usuario no tiene autorización para realizar esta acción.

Actores: Administrador

------------------------------------------------------------------------

## CU 3.4

Requisito funcional: Como cliente podré consultar mis cuentas bancarias.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema deberá devolver una lista de todas las cuentas bancarias asociadas al cliente autenticado.

Descripción: Un cliente puede obtener una lista de todas sus cuentas bancarias.

GET /vivesbank/v1/accounts/me

Secuencia normal:

1. El cliente envía una petición al sistema.

2. El sistema verifica que el usuario tenga un token de autenticación válido.

3. El sistema busca en la base de datos todas las cuentas bancarias asociadas al cliente autenticado.

4. El sistema devuelve una lista de las cuentas encontradas en la respuesta con un código de estado 200.

Secuencia alternativa:

2.1 El token de autenticación no es válido.

\* El sistema devuelve un mensaje de error con un código de estado 401 (Unauthorized), indicando que el usuario no está autorizado.

Actores: Cliente

---------------------------------------------------------------------------

## CU 3.5

Requisito funcional: Como cliente, podré crear una nueva cuenta bancaria.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema deberá crear una nueva cuenta bancaria y asignarle un número de cuenta único.

Descripción: Un cliente puede solicitar la creación de una nueva cuenta bancaria proporcionando los datos necesarios.

GET /vivesbank/v1/accounts

Secuencia normal:

1. El cliente envía una petición al sistema con los datos de la nueva cuenta.

2. El sistema valida los datos de la solicitud.

3. Si los datos son válidos, el sistema crea una nueva cuenta en la base de datos y asigna un número de cuenta.

4. El sistema devuelve una respuesta 201 (Created) con el número de cuenta de la nueva cuenta.

Secuencia alternativa:

2.1 Los datos de la solicitud no son válidos.

\* El sistema devuelve una respuesta 400 (Bad Request) indicando los errores encontrados en los datos.

Actores: Cliente

---------------------------------------------------------------------------

## CU 3.6

Requisito funcional: Como administrador podré eliminar una cuenta bancaria específica.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo y rol de administrador.

Postcondición: El sistema deberá eliminar la cuenta bancaria especificada, o devolver un mensaje de error si la cuenta no existe o no puede ser eliminada.

Descripción: Un administrador puede eliminar una cuenta bancaria específica proporcionando su ID.

DELETE /vivesbank/v1/accounts/{id}

Secuencia normal:

1. El administrador envía una petición al sistema incluyendo el ID de la cuenta que desea eliminar.

2. El sistema verifica que el usuario tenga el rol de administrador y que el token de autenticación sea válido.

3. El sistema busca en la base de datos la cuenta con el ID proporcionado.

4. Si la cuenta es encontrada y puede ser eliminada, el sistema la elimina de la base de datos.

5. El sistema devuelve un mensaje de confirmación con un código de estado 200.

Secuencia alternativa:

2.1 El usuario no tiene el rol de administrador o el token de autenticación no es válido.

\* El sistema devuelve un mensaje de error con un código de estado 403 (Forbidden), indicando que el usuario no tiene autorización para realizar esta acción.

3.1 La cuenta no es encontrada o no puede ser eliminada.

\* El sistema devuelve un mensaje de error con un código de estado 404 (Not Found).

Actores: Administrador

-----------------------------------------------------------------------------

## CU 3.7

Requisito funcional: Como cliente podré eliminar una de mis cuentas bancarias.

Precondición: Estar registrado en el sistema y tener un token de autenticación activo.

Postcondición: El sistema deberá eliminar la cuenta bancaria especificada, o devolver un mensaje de error si la cuenta no existe o no puede ser eliminada.

Descripción: Un cliente puede eliminar una de sus cuentas bancarias proporcionando su ID.

DELETE /vivesbank/v1/me/{id}

Secuencia normal:

1. El cliente envía una petición al sistema incluyendo el ID de la cuenta que desea eliminar.

2. El sistema verifica que el usuario tenga un token de autenticación válido.

3. El sistema verifica que la cuenta a eliminar pertenezca al cliente autenticado.

4. Si la cuenta es encontrada y puede ser eliminada, el sistema la elimina de la base de datos.

5. El sistema devuelve un mensaje de confirmación con un código de estado 200.

Secuencia alternativa:

2.1 El token de autenticación no es válido.

\* El sistema devuelve un mensaje de error con un código de estado 401, indicando que el usuario no está autorizado.

3.1 La cuenta no es encontrada o no puede ser eliminada.

\* El sistema devuelve un mensaje de error con un código de estado 404, indicando que la cuenta no existe o que no puede ser eliminada debido a restricciones.

Actores: Cliente

# Costes del proyecto

## Costes de personal:

Teniendo en cuenta que nos han contratado para un mes de trabajo se nos pagará por cada día trabajado.

Partimos de un sueldo base de 50€ al día, con un plus de productividad de 4€. Teniendo derecho a una paga extra con importe de 35 días de salario base. Con un IRPF del 11%  
  
Base de contingencias comunes:  
Remuneración = 50 + 4 = 54€

Prorrata de pagas extras -> (35\*50)/365 = 4,79€  
Base de contingencias comunes diaria = 54 + 4,79 = 58,79€

Base de contingencias comunes mensual = 58,79 \* 30 = 1763,7€  
  
Bruto = 1674€  
  
Contingencias comunes:   
1763,7€ \* 4,80% = 84,65€  
Desempleo:  
1763,7€ \* 1,55% = 27,33€  
Formación Profesional  
1763,7€ \* 0,1% = 1,76  
IRPF:  
1674 \* 11% = 184,14

Total: 1376,12€ netos

Teniendo en cuenta que somos 5 integrantes del grupo daría un resultado de 6880,6€ netos y 8370€ brutos.

## Costes de equipo:

Necesidad de 5 equipos portátiles para el desarrollo de la aplicación:

Portátil ASUS Vivobook 15 M1502YA-NJ448 AMD Ryzen 7 7730U/16GB/512GB SSD/15.6" – 529€  
Total = 2645€

Software:  
Licencia de jetBrains All products pack para organización con pago mensual – 77.90€

Total: 2645€ + 77.90€ = 2.722,9€

# Total de costes de proyecto = 11.092,9€

1. **Autores.**

### <https://github.com/Javierhvicente>

### <https://github.com/rraul10>

### <https://github.com/Samuceese>

### <https://github.com/javi97ruiz>

### <https://github.com/13elhad>

### 

**Fin.**

