MEMORIA DEL PROYECTO

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CERTIFICADO DE LOS TUTORES

D. Rodrigo Santamaría Vicente y D. Gabriel Villarubia González, profesores del Departamento de Informática y Automática de la Universidad de Salamanca.

Hacen constar:

Que el trabajo titulado "Sistema monetario y administración pública basado en blockchain" ha sido realizado por D. Felipe Sánchez Calzada con numero de documento 70905253-W y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado de la Titulación "Grado de Ingeniería Informática" de esta Universidad.

Y para que así conste a todos los efectos oportunos.

En Salamanca a 1 de Diciembre de 2021.

Firmado por SANTAMARIA VICENTE RODRIGO - 70879303L el día 10/01/2022 con un certificado emitido por AC FNMT Usuarios

VILLARRUBIA GONZALEZ GABRIEL - 71932845T

Firmado digitalmente por VILLARRUBIA GONZALEZ GABRIEL - 71932845T
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-71932845T, givenName=GABRIEL, sn=VILLARRUBIA GONZALEZ, cn=VILLARRUBIA GONZALEZ GABRIEL - 71932845T
Fecha: 2022.01.10 11:01:46 +01'00'

D. Rodrigo Santamaría Vicente **D. Gabriel Villarubia González**

RESUMEN

Durante los últimos años las criptomonedas han sido un tema recurrente en multitud de entornos, desde los más técnicos a los meramente informativos.

Lo que interesa desde el punto de vista técnico no son las criptomonedas, es la tecnología que hay detrás: la tecnología blockchain. Esta tecnología permite, gracias a su naturaleza, tener un histórico irrefutable de la base de datos no relacional distribuida definida.

Con este fondo, se decide desarrollar una plataforma que hospede un sistema monetario, que no una criptomoneda. Esta plataforma tiene dos puntos principales de interés. El primero, dejar de depender de los sistemas informáticos de los bancos para poder usar dinero digital. Y el segundo, automatizar tareas que han de hacerse regularmente con la moneda, como por ejemplo el pago de impuestos en facturas.

La plataforma, de ahora en adelante "NEuro", utiliza el sistema blockchain empresarial Hyperledger Fabric, mantenido por The Linux Foundation.

NEuro además posee una API REST publica que funciona como middleware, o capa media, para poder operar con la blockchain fácilmente y sin necesidad de conocer los detalles del funcionamiento de HL Fabric.

Para manejar las identidades y los roles de los usuarios, administradores, empresas y haciendas públicas, se utiliza Keycloak con la autenticación TOTP (Time-based One-Time Password) obligatoria. Esta característica aporta una capa de seguridad mejor incluso que el 2FA (Second Factor Authentication) por SMS.

Utilizando la API REST se ofrece un Frontend Web elaborado en Angular que permite a los usuarios operar con la moneda.

En la web los usuarios pueden operar según su rol:

- Administrador de identidades: Puede crear usuarios de tipo usuario y de tipo empresa.
- Usuario: puede crear cuentas bancarias, transferir fondos o pagar facturas.
- Empresa: puede hacer lo mismo que el usuario y además emitir facturas.
- Hacienda pública: puede hacer lo mismo que el usuario pero además una de sus cuentas bancarias se asigna para pagos de impuestos en los pagos de facturas.
- Super administrador: Puede crear administradores de identidades. Este rol se omite en cualquier web y es solo accesible desde la API REST.

Dado que NEuro está enfocado para ser usado por un gran número de usuarios todos los productos se ejecutan en contenedores Docker sobre Kubernetes y con multitud de replicas (A excepción de la blockchain que tiene su propio entorno). Esto permite escalar rápidamente en función de la carga.

De cara a la evolución rápida y solida del sistema en cuanto a funcionalidades, se ha decidido seguir las metodologías Agiles para la gestión del proyecto.

NEuro consta además de automatizaciones CI/CD (Continuous Integration and Continuous Delivery) para poder suplir de forma rápida las necesidades del proyecto.

Palabras clave: Blockchain, Sistema Monetario, Moneda, Hacienda Pública, Kubernetes, Keycloak, Hyperledger Fabric, Springboot.

SUMMARY

During the last few years cryptocurrencies have been a recurring topic in a multitude of environments, from the most technical to the merely informative.

What is of interest from a technical point of view is not cryptocurrencies, but the technology behind them: blockchain technology. This technology allows, thanks to its nature, to have an irrefutable history of the defined distributed non-relational database.

With this background, it is decided to develop a platform that hosts a monetary system, not a cryptocurrency. This platform has two main points of interest. The first is to stop relying on banks' computer systems to be able to use digital money. And the second is to automate tasks that have to be done regularly with the currency, such as paying taxes on invoices.

The platform, hereafter "NEuro", uses the Hyperledger Fabric enterprise blockchain system, maintained by The Linux Foundation.

Neuro also has a public REST API developed in Java Springboot that works as middleware, or middle layer, to be able to operate with the blockchain easily and without the need to know the details of how HL Fabric works.

To manage the identities and roles of users, administrators, companies and treasuries, Keycloak is used with mandatory TOTP (Time-based One-Time Password) authentication. This feature provides a layer of security even better than 2FA (Second Factor Authentication) via SMS.

Using the REST API a Web Frontend elaborated in Angular is provided that allows users to trade with the currency.

On the web users can trade according to their role:

- Identity Manager: Can create user type and company type users.
- User: can create bank accounts, transfer funds or pay bills.
- Business: can do the same as the user and also issue invoices.
- Tax Athority: can do the same as the user, but in addition one of their bank accounts is assigned for tax payments on invoice payments.
- Super administrator: You can create identity administrators.

Since NEuro is focused to be used by a large number of users all products run in Docker containers on top of Kubernetes and with a multitude of replicas (Except for the blockchain that has its own environment). This allows to scale quickly depending on the load.

In order to ensure a fast and solid evolution of the system in terms of functionalities, it has been decided to follow Agile methodologies for project management.

NEuro also includes CI/CD (Continuous Integration and Continuous Delivery) automations in order to quickly meet the needs of the project.

Keywords: Blockchain, Monetary System, Currency, Treasury, Kubernetes, Keycloak, Hyperledger Fabric, Springboot.

CONTENIDO

1 DESCRIPCIÓN DE LOS ANEXOS Y ENTREGA

En los anexos se describe con detalle cada punto técnico del sistema. Junto con los anexos se entrega también el código fuente de todo el sistema, que va provisto de un control de versiones Git con el que se puede ver el avance por fechas.

Es muy importante tener en cuenta que los anexos son una parte clave de este proyecto y no han de considerarse ampliación de la Memoria (el presente documento). Este documento no tiene en sí mismo la información necesaria para comprender el sistema y será necesario leer en orden los Anexos para ir entendiendo el funcionamiento y las claves de proyecto.

Descripción de anexos:

- **Anexo I – Arquitectura:** Se define la arquitectura de todo el sistema desde el punto de vista de la integración y comunicación de los diferentes componentes individuales.
- **Anexo II – Software y estructura de datos:** en este anexo se explican ciertos detalles del software, sin entrar en la arquitectura, que se define en el anexo anterior. En él se reflejan ejemplos de código relevantes. También añade la estructura de datos que la blockchain almacena.
- **Anexo III – Infraestructura:** Descripción detallada de los servicios usados para el despliegue de los diferentes componentes en producción. También se describe un entorno ideal junto con una aproximación de costos de ese entorno.
- **Anexo IV – Monitoreo:** Herramientas usadas para el monitoreo del sistema desplegado en producción.
- **Anexo V – CI-CD:** Detalles sobre los procesos CI/CD usados durante el desarrollo.
- **Anexo VI – Requisitos y planificación:** Se describen las necesidades del negocio y los requisitos obtenidos de estas. También se define como se ha estimado la temporalidad de los requisitos.
- **Anexo VII – Manuales:** Manual detallado de la aplicación web. Sirve también como pequeña muestra de la interfaz.

2 INTRODUCCIÓN

En este y en los documentos anexos (muy importantes) se describe con detalle el funcionamiento y los rasgos técnicos del sistema NEuro. Este sistema ha sido desarrollado como Trabajo de Fin de Grado para el Grado en ingeniería informática impartido por la Universidad de Salamanca.

Actualmente todos los bancos tienen su propio software para manejar dinero digital. Básicamente, si quieres poder pagar digitalmente, tienes que firmar un contrato con un banco, que lo más común es que sea privado y tenga sus condiciones de uso sujetas al contrato.

Ante esta situación, mucha gente se ve obligada a contratar los servicios de bancos privados cuando lo único que quieren es operar digitalmente y de forma gratuita.

Pues bien, dándole muchas vueltas a esta, y otras cuestiones, me surgen las ideas principales del proyecto:

- Que el banco central responsable de una moneda sea el que da la opción de manejar dinero digital
- Que este mismo banco central proporcione la posibilidad de que los países asociados automaticen tareas como el pago de impuestos. Al final pagar una factura no es más que hacer una transferencia de fondos a la empresa y pagar los impuestos relacionados con esa factura.

Otro de los puntos clave de NEuro es dar la posibilidad de que un banco privado se una al sistema, no solo interactuando con él, si no siendo parte de él. Esto en el leguaje técnico se va a traducir en que la organización Hyperledger Fabric del banco privado se una al Cannel que la Organización el banco central tiene, y que así ejecute los Contratos Inteligentes del Channel que el banco central creó.

3 OBJETIVOS

Como se menciona anteriormente el objetivo principal es permitir a cualquier persona que usa un determinado sistema monetario disponer de dinero digital sin necesidad de contratar un banco privado.

3.1 OBJETIVOS FUNCIONALES

Estos objetivos son los que darán lugar a las diferentes funcionalidades del sistema:

- Posibilidad de las personas de tener sus cuentas bancarias en el sistema
- Transferir fondos entre cuentas bancarias
- Generación de facturas desde las empresas
- Pago de facturas ya sea desde empresa o desde usuario
- Pago automático de impuestos al pagar facturas
- Posibilidad de que las haciendas públicas tengan sus cuentas bancarias para recibir los pagos de impuestos

- Registro de todas las transacciones o modificaciones que escriban nuevos datos, sean del carácter que sean (Transferencias, generación de facturas, creación de cuentas bancarias…). Para la lectura no es necesario almacenar ningún registro.
- Posibilidad de traducción del sistema (Aunque solo se implemente un idioma)

3.2 OBJETIVOS NO FUNCIONALES

Los objetivos no funcionales, en este caso, va a ser aquellos que no dan lugar a ninguna funcionalidad en el sistema.

- Despliegue de todos los subsistemas, a excepción de la blockchain en el un entorno de contenedores. En concreto Kubernetes u OpenShift.
- Despliegue del sistema en servidores propios (Baremetal).
- Dado que se desplegará en servidores propios, también se configurarán todos los sistemas, clústeres y redes.
- Tener el sistema desplegado en un entorno productivo bajo el dominio neuro-front.felipesanchezweb.es
- Definición de arquitectura robusta y enfocada a microservicios y micro aplicaciones, aunque en esta interacción del proyecto solo exista un servicio.
- Autenticación de usuarios segura obligatoria.

3.3 OBJETIVOS PERSONALES

El objetivo principal que tenía en mente al realizar este proyecto era probar mi capacidad para aprender tecnologías muy muy nuevas y complejas y por lo tanto con poca documentación como es la blockchain Hyperledger Fabric. Con poca documentación no me refiero a la oficial, que es bastante buena, si no a la que se va elaborando por la comunidad (Stack Overflox, Youtube, etc.)

Otro de los objetivos importante es desarrollar un sistema moderno con capacidad de despliegue en entornos de contenedores y con una alta escalabilidad.

Comprobar la dificultad y el costo de desarrollar un proyecto de manera individual sin un equipo.

La carrera enseña multitud de conceptos, pero todos ellos normalmente separados, con este proyecto también he pretendido sintetizarlos, unirlos y hacerlos visibles en un producto completo y real.

Dado que llevo ya más de un año trabajando en el sector en consultoras de software, otro objetivo ha sido saber aplicar las técnicas que las empresas enseñan y que dan un paso más en la formación que la universidad da.

Por último, y aunque parezca que va en contra de la gestión de proyectos en general, he querido ir cambiando los requisitos durante el desarrollo. Esto en un mundo ideal no debería de pasar, pero en el mundo real pasa siempre. No hay un solo proyecto que desde su inicio a su fin tenga exactamente los mismos requisitos funcionales sin cambios. Es por ello por lo que he querido poner el objetivo de "adaptarme a los cambios" que el negocio exige.

4 TÉCNICAS Y HERRAMIENTAS

En esta sección se describen las diferentes tecnologías, lenguajes, herramientas, etc. que se utilizan para desarrollar y poner en marcha el sistema NEuro.

Las herramientas descritas según su pertenencia a los módulos y si alguna de ellas se repite simplemente se nombrará.

En esta sección además se aclarará el motivo de la utilización, y donde se ha usado. Para entender profundamente el uso de cada herramienta es necesario acudir a los anexos.

A modo de introducción a continuación se encuentra una imagen en la que se puede ver la composición y comunicación de los diferentes módulos.

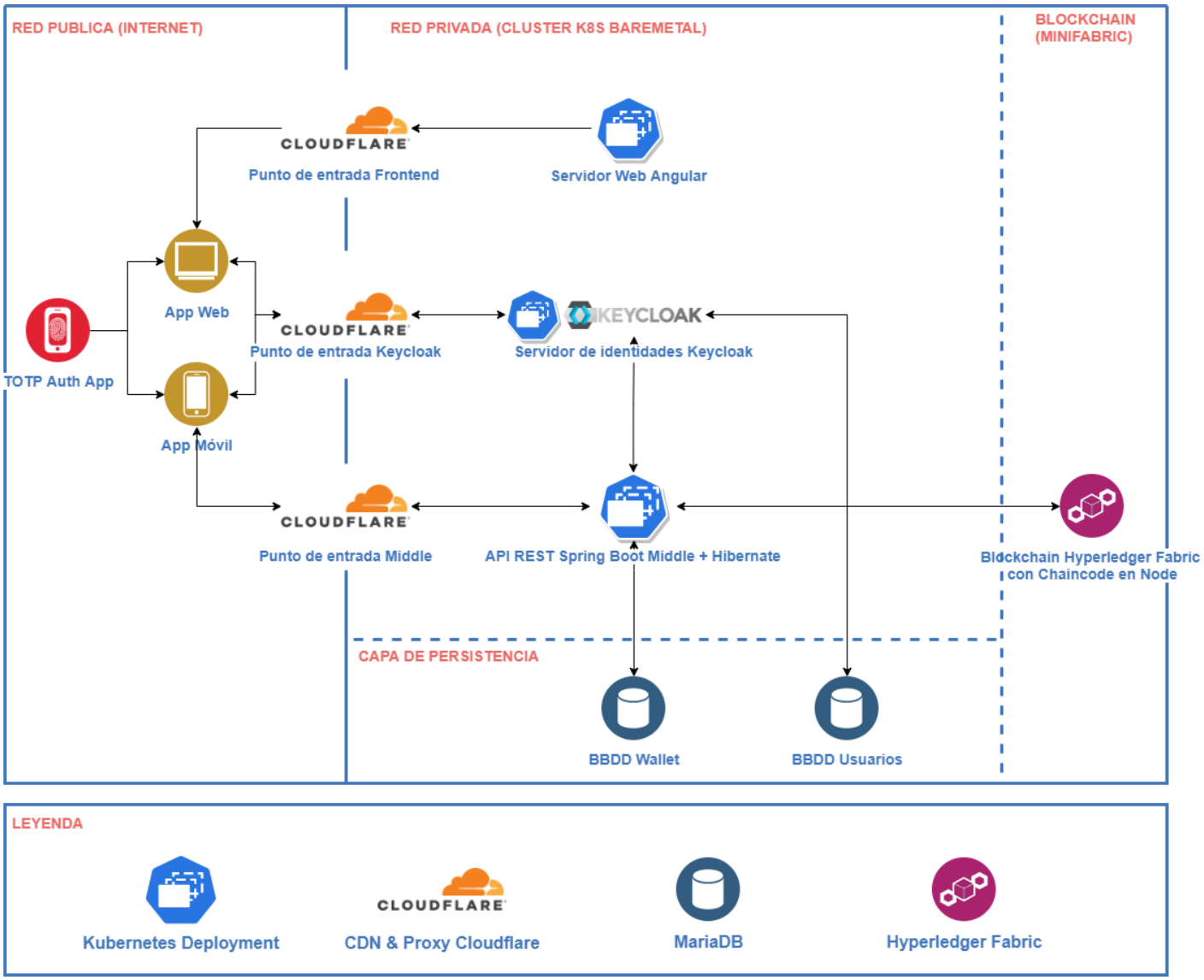


Ilustración 1: Arquitectura resumida del sistema NEuro

En el "Anexo I – Arquitectura" se describirá a detalle la arquitectura del sistema.

4.1 ENTORNO DE DESARROLLO

4.1.1 VS CODE

Visual Studio Code es un editor de código ofrecido por Microsoft. Lo que hace interesante a este editor es la cantidad de plugins, que pueden convertir el editor en un completo Entorno IDE (Entorno de Desarrollo Integrado, Integrated Development Environment).

En NEuro, VS Code es utilizado para el desarrollo de los Snart Contracts (Contratos Inteligentes o Chaincode: es la denominación que se le da a los programas que se ejecutan dentro de la blockchain).

La decisión ha sido tomada por la existencia del plugin o extensión "IBM Blockchain Platform" que convierte el editor en un IDE Perfecto para desarrollar Contratos Cnteligentes en Hyperledger Fabric.

4.1.1.1 IBM BLOCKCHAIN PLATFORM PLUGIN

Este plugin, como ya he introduje anteriormente, convierte VS Code en un completo IDE específico para desarrollo de Smart Contracts en Hyperledger Fabric.

Hasta el momento es la forma más rápida y eficiente de desarrollar contratos inteligentes en Node (Javascript en el lado del servidor)

Las características específicas que he usado son la siguientes (Aunque tiene muchas más):

- **Creación automática de una red HL Fabric:** Nos permite crear una red en nuestro Docker local haciendo un simple click. Podemos exportar los perfiles de conexión para la red pudiendo así usarla en los SDK (Software Development Kit), en NEuro concretamente el SDK de Java.
- **Depuración de Smart Contracts:** con la red que podemos crear desde el plugin podemos ejecutar los Smart Contracts en modo de depuración. Con ello es posible establecer puntos de ruptura y ver exactamente el valor que toman las variables.
- **Ejecución de transacciones sobre los Smart Contracts:** esta característica va de la mano con la depuración. Podemos ejecutar las transacciones desde el plugin sin necesidad de conectarnos con un SDK a la red.

Aunque es una muy buena extensión, no es perfecta. Algunos de los puntos que he debido tener en cuenta:

- Ya es estable, pero sigue en pleno desarrollo, lo que trae con cada versión nuevas características, pero la eliminación de otras, lo que trae problemas de compatibilidad con versiones anteriores.
- Actualizar es peligroso. Con las últimas versiones eliminan la posibilidad de depurar código. Además, cambian completamente la forma en la que se despliega la red HL Fabric local, lo que hace que la depuración no funcione como antes.
- Aun no se pueden depurar Smart Contracts en la versión 2.0 de HL Fabric.

4.1.2 INTELLIJ IDEA

Desarrollado por JetBrains, es uno de los mejores IDEs para desarrollo de Java. Tanto es así que el propio Android Studio, IDE de referencia en desarrollo Android, esta basado en IntelliJ.

Tiene una amplia gama de extensiones, aunque no es necesario instalar ninguna, no se echa nada en falta en este IDE.

Se ha usado para desarrollar el Middle Java Springboot.

De este IDE vienen todos los de la familia de JetBrains, cada uno específico para desarrollar en un conjunto de lenguajes.

4.1.3 WEBSTORM

Se trata del IDE específico para desarrollo Web de JetBrains. Es idéntico a IntelliJ pero con las características específicas necesarias para desarrollo Web.

Se ha utilizado para el desarrollo del Frontend Angular.

4.1.4 POSTMAN

Esta herramienta permite hacer llamadas a APIs REST.

Es interesante porque permite guardar las llamadas, crear carpetas de llamadas, variables en las llamadas y otras funciones específicas.

En el proyecto se ha utilizado para probar el funcionamiento del Middle antes de añadir los Endpints (las URLs de un API o un backend) funcionando en el Frontend.

4.2 BLOCKCHAIN

Este módulo corresponde al almacenamiento blockchain que utiliza NEuro, es al fin y al cabo el punto final de cualquier transacción.

4.2.1 JAVASCRIPT

Javascript es un lenguaje de programación de propósito general con compilación Just-in-time (JIT). En algunas bibliografías se denomina "lenguaje de programación interpretado", que también es correcto.

Es orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

4.2.2 TYPESCRIPT

Se define como JavaScript tipado a cualquier escala (Typed JavaScript at Any Scale).

En NEuro es el lenguaje de programación utilizado para los Smart Contracts y para el Frontend.

Motivos de la elección:

- Es el lenguaje usado en Angular por lo que las interfaces y clases pueden ser compartidas por los Smart Contracts y por el Frontend, de esta manera se evita definir los mismos tipos en varios lenguajes.
- Es muy muy sencillo y trae todas las funciones por defecto de JavaScript.
- Hay bibliotecas prácticamente para todo debido a la gran comunidad. Usando NodeJS tiene el registro de software mas grande del mundo.
- Puedes usar bibliotecas TypeScript o JavaScript.

4.2.3 NODEJS

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.

Permite ejecutar JavaScript (Y por tanto TypeScript) en otros entornos diferentes de un navegador.

Hyperledger Fabric permite ejecutar Chaincode en este entorno.

4.2.3.1 NPM

Es el gestor de paquetes por defecto para NodeJS.

Es el registro de software (en inglés Registry, lugar en el que se almacenan las librerías y paquetes) más grande del mundo.

Dispone de CLI (Command Line Interface), sitio web y el registro.

4.2.4 HYPERLEDGER FABRI

Hyperledger Fabric es un proyecto Open Source (De código abierto) mantenido por la Linux Fundation. Se trata de una blockchain modular que es considerada el estándar en blockchains empresariales.

Proporciona una blockchain permisionada y privada.

Una blockchain permisionada privada es aquella en la que los usuarios no son anónimos y necesitan identificarse. Algunas de sus características son las siguientes:

- El acceso está restringido a aquellos que una unidad central permite. En el caso de HL Fabric se denominan CAs (Certificate Authority, Autoridad de Certificacion)
- Las transacciones y el acceso a datos son privado.
- Normalmente no existen mineros si no que son simples nodos los que aprueban las transacciones. En HL Fabric se denominan Peers.

4.2.4.1SMART CONTRACTS O CHAINCODE

Se denomina Smart Contracts a los programas que se ejecutan en la blockchain.

En el caso de Hyperledger Fabric disponemos de SDKs para elaborar Chaincode en Go, NodeJS (JavaScript) y Java. La elección como ya he descrito anteriormente ha sido NodeJS con TypeScript.

4.3 MIDDLE

El middle se encarga de:

- Almacenar los certificados digitales de cada usuario en una base de datos privada. Esta base de datos es MariaDB (Versión libre de MySQL).
- Autenticar a los usuarios contra el servidor de autenticación Keycloak
- Escuchar las llamadas REST para ejecutar las transacciones correspondientes contra HL Fabric. Actuaría como el cliente de HL Fabric.

Está escrito en Java.

4.3.1 MARIADB / MYSQL

Como mencionaba antes, MariaDB es la versión libre de MySQL.

MySQL es un sistema gestor de bases de datos relacionales altamente utilizado en la industria. Es mantenido por Oracle.

4.3.2 JAVA SPRINGBOOT Y SPRING FRAMEWORK

Spring Framework es un Framework de desarrollo Web escrito en Java. Es complejo de configurar y los compilados generados son WAR y no JARS, por lo que necesitan un servidor web para ejecutarse.

Spring Boot simplifica el proceso de configurar y compilar aplicaciones Spring Framework al máximo. Dispone de dos elementos que hacen el desarrollo bastante amigable:

- **Contenedor integrado de aplicaciones:** permite compilar la aplicación en un JAR y desplegarla como una aplicación java normal. Para ello integra en su propio servidor de aplicaciones Tomcat. Esta característica permite desplegar las aplicaciones en contenedores Docker o similares.
- **Starters:** Con ellos vamos a poder incorporar fácilmente funcionalidades como conectores a bases de datos, a servidores de colas, etc. En NEuro se conecta a MariaDB a través de JDBC.

4.3.3 JAVA HIBERNATE

Se trata de un ORM (Object-Relational mapping) para JAVA que sirve para mapear los atributos de los modelos relacionales (MySQL, PostgreSQL, etc) con objetos Java.

En NEuro simplemente se utiliza para almacenar y recuperar las Wallets HL Fabric de la base de datos desde el Middle.

4.4 FRONTEND

El frontend, como su nombre indica, es la parte frontal, es decir, la parte con la que los usuarios van a interactuar.

En el caso de NEuro, se trata de una Web SPA (Single Page Application). Este tipo de webs solo tienen una página, por lo que la web solo se carga una vez y los datos se van cargando dinámicamente cuando se necesitan.

Elaborar Webs SPA sin un Framework es una tarea lenta, compleja, propensa a errores.

4.4.1 ANGULAR

Dado que es inviable desarrollar una SPA sin un Framework, es aquí donde hay que decidir cual elegir (los tres son libres).

- React: creado y mantenido por Facebook. Se trata de una biblioteca y no un Framework. Es el más usado.
- Vue: creado por Evan You y mantenido por la comunidad. Muy ligero.
- Angular: Creado y mantenido por Google. Pesado pero muy muy completo.

Podríamos estar hablando largo y tendido sobre cual elegir, y en realidad no es más que una elección personal. Los motivos por los que he elegido Angular sobre React y Vue:

- **Proporciona todo lo necesario** y lo que puedas necesitar en un futuro, desde servicios, interceptores, enrutadores, hasta guardianes de ruta, etc. A diferencia del resto, que dejan en manos de terceros algunas funcionalidades, con Angular no es necesario recurrir a terceros. Este punto es ideal para mí, que el propio Framework proporcione estas características hace que estén diseñadas según los patrones de diseño del Framework, haciendo que este perfectamente unido con el.
- **Desarrollado en TypeScript**
- **Proporciona una arquitectura predefinida**, y bastante limpia:
- **Tiene su propio lenguaje de plantillas**
- **Es muy adecuado para aplicaciones grandes:** gracias a la arquitectura que proporciona.

En realidad, el punto de más peso es que proporcione todo lo que necesita cualquier sistema sin tener que recurrir a terceros.

Uno de los puntos negativos, y por lo que mucha gente lo declina es que es complejo, y hay que tener conocimientos bastante avanzados para empezar a hacer las cosas bien (Curva de aprendizaje muy pronunciada). Para mi eso no supone ningún problema.

4.4.2 PRIMENG

PrimeNG es una de las bibliotecas de componentes para Angular mas grandes que existen. Es mantenida por PrimeTek Informatics y es libre.

En el caso de NEuro, se ha utilizado un tema de pago, se trata del tema Freya y su precio ronda los 59 dólares para uso no comercial, 590 para uso comercial.

La razón principal para usar una biblioteca de componentes es que simplifica mucho el trabajo de diseño y maquetación. Si en el equipo de desarrollo no hay un diseñador, se hace imprescindible usar una biblioteca hecha por expertos en diseño para obtener un resultado profesional, y por qué no, bonito.

Otras razones para usar PrimeNG:

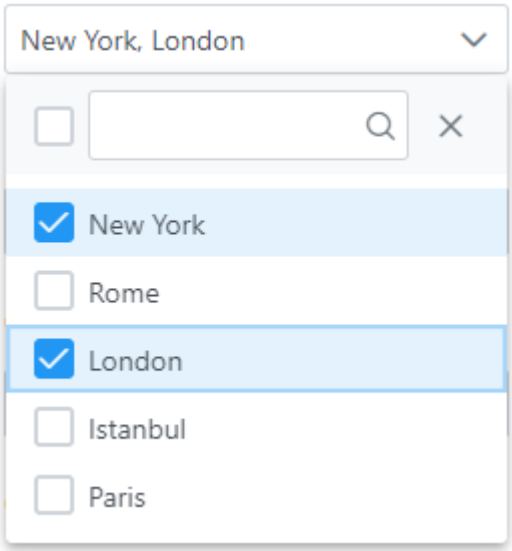
- **Sus componentes son de altísima calidad y muy personalizables.**
- **Dispone de componentes para prácticamente cualquier funcionalidad:** otras bibliotecas como Angular Material (la biblioteca de referencia para Angular) tienen fuertes carencias en cuanto a la cantidad de componentes.
- **No requiere uso de componentes de terceros:** esto es vital y permite customizar los temas en tiempo real de manera limpia y sin hacer adaptaciones.

El mayor punto negativo es la documentación de su API, que se basa principalmente en ejemplos, y algunas de las funciones no están documentadas. Aun así, hay muchísimos y muy buenos ejemplos de uso de cada componente.

Un pequeño ejemplo de uso de uno de sus componentes:

```
<p-multiSelect [options]="cities" [(ngModel)]="selectedCities1"
    defaultLabel="Select a City" optionLabel="name"
    selectedItemsLabel="{0} items selected">
</p-multiSelect>
```

El fragmento de código da lugar a este componente



Como vemos, con muy poco código podemos hacer muchísima funcionalidad. En este caso un selector múltiple con buscador integrado.

4.5 AUTENTICACIÓN

En NEuro el sistema de autenticación es totalmente externo al resto de servicios. Esto permite añadir mas módulos o servicios sin necesidad de adaptar los demás para soportar la autenticación.

Se utiliza Keycloak como servidor de autenticación.

4.5.1 KEYCLOAK

Se define como "Open Source Identity and Access Management", es decir, Administración de identidades y accesos open source.

Es mantenido por RedHat y tiene algunos puntos clave que lo hacen ideal para NEuro:

- Sporte SSO
- Soporte LDAP y Active Directory
- Alto rendimiento
- Soporte OpenID y SAML

En NEuro se utiliza tanto en Angular como Springboot con el protocolo OpenID

4.5.2 OPENID

Se define como un estándar de identificación digital descentralizado.

4.5.3 TOTP

TOTP proporciona un segundo factor de identificación basado en tiempo. Es obligatorio para cualquier usuario en NEuro.

En primer lugar, hay que definir que es OTP (One Time Password): como su nombre indica, no es más que una contraseña de un solo uso. El punto clave está en cómo se generan esas contraseñas de un solo uso y si se combinan o no con contraseñas corrientes.

TOTP (Time-based One Time Password): Se trata de una contraseña de único uso basada en tiempo.

Su funcionamiento es simple: se tiene una semilla estática que tanto el generador de la contraseña como el comprobador conocen. El generador generará una contraseña nueva válida para un intervalo de tiempo (por ejemplo 30 segundos) y el comprobador comprueba que la contraseña coincide con la generada por él mismo.

Este tipo de autenticación es bastante seguro, mas incluso que el OTP en el que se recibe la contraseña por SMS, aunque es vulnerable a ataques pishing y ataques en los que se roban las semillas.

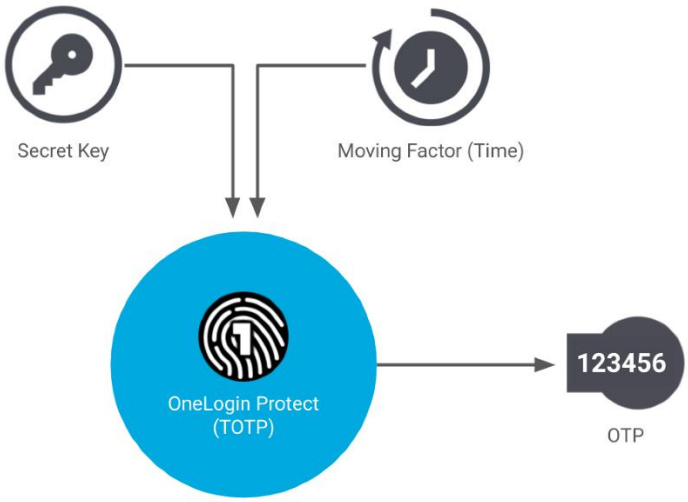


Ilustración 2: Funcionamiento ilustrativo de TOTP

El generador de contraseñas TOTP es normalmente un dispositivo móvil con una aplicación, por ejemplo, Google Authenticator.

Hay veces que TOTP no funciona bien debido a desfases temporales entre el cliente y el servidor. Para ello se puede configurar una ventana de tiempo en la que el toquen sigue siendo válido aun estando caducado. Esto en Keycloak únicamente soluciona desfases temporales en una dirección, asique no es perfecto

También es posible usar HOTP en casos en los que la hora no se puede garantizar.

En NEuro, Keycloak se encarga de esto.

4.5.4 SSO

SSO (Single Sign-On) es una forma de autenticación que permite tener acceso a varios sistemas.

La forma clásica de inicio de sesión o autenticación da acceso al sistema en el que inicias la sesión. Con SSO la sesión se guarda en el cliente y se utiliza para varios sistemas.

OpenID (Ya descrito anteriormente) es un tipo de SSO distribuido.

4.5.5 TOKEN JWT

JWT (JSON Web Tokens) es un estándar abierto (RFC 7519) para representar información JSON de forma segura entre pares.

Estos tokens contienen una firma, haciendo inmutable la información que almacenan.

Por ejemplo, este token (No contiene saltos de línea, simplemente no entra en el documento):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 .
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKK
F2QT4fwpMeJf36POk6yJV_adQssw5c
```

Contiene la siguiente información:

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
  }
```

Los JWT tienen tres partes separadas por punto (carácter “.”). La primera corresponde a la cabecera, la segunda es la carga útil y la tercera es la firma.

Es muy utilizado para almacenar información de inicios de sesión y autenticación.

NEuro los utiliza para guardar la información en el navegador del usuario autenticado.

4.5.5.1 JSON

JSON (JavaScript Object Notation) es un formato para representar datos.

Un ejemplo puede ser el siguiente:

```
{
    "nombre": "Felipe",
    "apellidos": [
        "Sanchez",
        "Calzada"
    ],
    "edad": 22,
    "fecha_nacimiento": {
        "dia": 25,
        "mes": 7,
        "anio": 1998
    }
}
```

4.6 DESPLIEGUE

Para NEuro se ha elaborado un despliegue completo, desde la instalación de los servidores hasta la instanciación de los clústeres de Kubernetes.

4.6.1 PROXMOX

Proxmox Virtual Environment es una plataforma de gestión para maquinas virtuales de código abierto. Mantenido por Proxmox Server Solutions GmbH Se basa en:

- **QEMU**: Emulador/virtualizador de máquinas opensource.
- **KVM**: Kernel-based Virtual Machine: solución para virtualización completa con Linux.
- **LXC:** LinuX Containers, permite que un servidor físico ejecute diferentes instancias de "espacios de nombres" (Jaulas o Entornos Virtuales).

Otras características destacables:

- Permite la creación de clústeres de alta disponibilidad
- Creación de redes virtuales de alto rendimiento
- Interfaz gráfica web bastante sencilla.

La alternativa que también se ha analizado es XenServer y su versión opensource actualizada, XCP-ng. Aunque el rendimiento y las funcionalidades son muy buenas, cuando se virtualiza un Firewall/Router PfSense las redes virtuales tienen un rendimiento bastante pobre e inestable, por lo que XenServer ha sido descartado desde el inicio de las pruebas de concepto.

Proxmox se ejecuta por encima de un sistema operativo, en este caso, encima de Debian, aunque ya trae un instalador que lo instala automáticamente.

4.6.2 DEBIAN

Debian es una distribución GNU/Linux universal opensource. Está disponible en compilaciones para múltiples arquitecturas.

Es, junto con Ubuntu (distribución basada en el propio Debian) la versión más popular.

Los motivos de usar Debian son: su gran comunidad, y que es la distribución con la que más cómodo me siento y más experiencia tengo. Además, es la versión que recomienda Proxmox usar y es soportada de forma nativa.

4.6.3 GITLAB

Se trata de un servidor para almacenamiento de repositorio Git. Se proporciona como SaaS (Software as a Service) o también se puede hacer una instalación en servidores gestionados propios, esta última es la opción usada en NEuro.

En realidad, actualmente es una suite completa para el desarrollo y gestión del ciclo de vida software con el código como parte principal.

Uno de los puntos más importantes son sus GitLab Workers, que permiten seguir técnicas DevOps para CI/CD (Integración Continua y Entrega Continua).

NEuro utiliza todas las características que proporciona para desplegar de forma autónoma y a partir de un commit de Git todo el sistema, a excepción de los Smart Contracts.

4.6.4 PFSENSE

Se define como "El firewall de código abierto más usado del mundo".

Se trata de un Firewall/Router basado en software de código abierto. Se ofrece como una distribución basada en FreeBSD con kernel personalizado.

También se ofrece con instalaciones en la nube o con su propio hardware, bajo la empresa Netgate® en diferentes configuraciones.

NEuro lo utiliza como Router/Firewall principal. Está instalado en una máquina virtual y con varias redes virtuales dentro de Proxmox.

4.6.4.1 HAPROXY

Se define como un equilibrador de carga TCP/HTTP confiable y de alto rendimiento. En realidad, es mucho más que eso y permite establecer reglas complejas basadas en TCP/HTTP.

PfSense lo ofrece como extensión instalable y se integra completamente haciendo la configuración mucho más sencilla a través del panel web de PfSense.

NEuro lo utiliza como proxy inverso para poder servir diferentes recursos a través de la misma IP y puerto.

4.6.4.2 ACME

Con esta extensión podemos gestionar certificados digitales para securizar los sitios que atraviesan nuestra instancia de PfSense.

Proporciona renovación automática de certificados, expedición desde diferentes plataformas y otras características.

NEuro lo utiliza junto con Cloudflare y LetsEncrypt para proporcionar los certificados internos.

4.6.5 MINIFABRIC

Minifabric es una herramienta libre que permite desplegar una red Hyperledger Fabric completa.

Con ella es posible desplegar pequeñas redes o redes de grandes dimensiones multi-maquina.

Internamente funciona con playbooks de Ansible (herramienta de orquestación y automatización).

El archivo de configuración que Minifabric recibe en NEuro es el siguiente:

```
fabric:
  cas:
    - ca1.neuroorg.neuro.com
  peers:
    - peer1.neuroorg.neuro.com
    - peer2.neuroorg.neuro.com
    - peer3.neuroorg.neuro.com
  orderers:
    - orderer.neuro.com
  settings:
    ca:
      FABRIC_LOGGING_SPEC: ERROR
    peer:
      FABRIC_LOGGING_SPEC: ERROR
    orderer:
      FABRIC_LOGGING_SPEC: ERROR
  netname: neuro-minifabric-net
  container_options: "--restart=always --log-opt max-size=1000m --log-
opt max-file=3"
```

El archivo es autodescriptivo, genera una CA, tres Peers y un Orderer.

Los comandos para desplegar, instalar e instanciar la red y los Smart contracts son los siguientes:

```
# minifab up -i 1.4.5 -c neurochannel -s couchdb -o neuroorg.neuro.com -e 10000
# minifab install -n neuro -v 0.0.1 -l node
# minifab instantiate -p ""
```

Las opciones usadas para el levantamiento de la red:

- **-i 1.4.5**: especifica la versión 1.4.5 de Hyperledger Fabric
- **-c neurochannel**: crea el Channel neurochannel
- **-s couchdb**: pone chouchdb como base de datos del Ledger (Libro mayor de Hyperledger Fabric)
- **-o neuroorg.neuro.com**: especifica el nombre de la organizacion

- **-e 10000**: establece el puerto inicial para los nodos, cada nodo irá aumentando en 1 el numero de puerto a partir del 10000
- **-n neuro**: nombre del chaincode
- **-v 0.0.1**: versión del chaincode
- **-l node**: plataforma del chaincode (node, java o go)
- **-p "”**: parámetros de ejecución de la función inicial (“” significa vacío, sin parámetros)

4.6.6 KUBERNETES

Abreviado como k8s, se trata de un orquestador de contenedores (normalmente Docker, aunque no necesariamente) open source. Fue liberado por Google en 2014.

Con el podemos crear replicas de servicios escalables de forma muy sencilla.

NEuro despliega todos sus servicios a excepción de la blockchain en un clúster Kubernetes. El cluster consta de las siguientes maquinas (Todas dentro de un HP Proliant con Proxmox):

- 1x Nodo maestro
- 2x Nodos Workers

La definición de la infraestructura se hace normalmente con archivos YAML

4.6.6.1K0S

Desplegar un clúster Kubernetes completo puede hacerse una tarea compleja y bastante monótona.

K0s facilita esta tarea haciendo que desplegar un cluster sea tan sencillo como ejecutar unos pocos comandos.

Desplegar un nodo master:

```
k0s install server
systemctl daemon-reload
systemctl enable k0sserver
systemctl start k0sserver
```

Desplegar un nodo Worker:

```
workerToken='token’

curl -sSLf k0s.sh | sh
k0s install worker
sed -
i "s#ExecStart=/usr/local/bin/k0s worker#ExecStart=/usr/local/bin/k0s wor
ker $workerToken#g" /etc/systemd/system/k0sworker.service
```

```
systemctl daemon-reload
systemctl enable k0sworker
systemctl start k0sworker
```

4.6.6.2 METALLB

MetalLB es un balanceador de carga para Kubernetes.

Por defecto Kubernetes no trae un balanceador como tal ya que cada plataforma cloud integra la suya.

En el caso de desplegar un clúster en servidores propios es necesario añadir un balanceador como MetalLB.

4.6.6.3LENS

Lens, "The Kubernetes IDE". Se trata de una herramienta open source con interfaz gráfica que nos permite hacer todo lo que necesitemos en nuestro clúster de Kubernetes.

Desde crear instancias hasta monitoreo de recursos.

También permite ejecutar comandos de la CLI kubectl (CLI por defecto para administrar clisteres k8s)

En NEuro con esta herramienta se monitorea todo excepto la blockchain y los servidores de bases de datos (MariaDB).

4.6.7 PORTAINER

Es una herramienta open source con GUI (Interfaz grafica) para administrar y monitorear contenedores, entre otras cosas.

NEuro la utiliza solamente para el monitoreo de la red Hyperledger Fabric

4.6.8 CLOUDFLARE

Cloudflare en sus inicios nació como un CDN (Content Delivery Network, red de distribución de contenidos). Actualmente es prácticamente un completo proveedor de servicios en la nube, y que además tiene las características avanzadas de CDN.

En NEuro se utiliza para:

- **DDNS:** DNS dinámico, dado que mi IP es dinámica, es necesario usar un servicio que actualice la IP en los DNI de forma dinámica a medida que va cambiando.
- **Proxy IP:** esta característica permite ocultar la IP real en la que se encuentra el servidor. Un cliente siempre se conecta a las IP que Cloudflare proporciona antes que a la del servidor.
- **Protección y analíticas DDosS:** los ataques DDoS o de denegación de servicio son ataques en los que se satura la capacidad de un sistema con mas peticiones de las que soporta. Cloudflare tiene un sistema de protección contra estos ataques que no requiere de configuración.
- **Certificados digitales:** aunque internamente se usan certificados digitales expedidos por LetsEncrypt, de manera externa se usan los que proporciona Cloudflare.

4.6.9 STATPING

Se trata de un proyecto OpenSource que expone una interfaz con datos sobre la disponibilidad de un determinado servicio.

Permite configurar multitud de formas para la obtención de disponibilidad.

En NEuro se utilizan llamadas REST simples para verificar si los servicios responden con un código 200 (Satisfactorio).

4.6.10 PUSHOVER

Se trata de un servicio de notificaciones push extremadamente simple.

Dispone de una APP para todas las plataformas y tiene un costo de 5$ de por vida.

Para mandar una notificación push a todos los dispositivos basta con llamar a un servicio REST con el token que se genera para tu cuenta en Pushover.

4.6.11 PROMETHEUS

Es una herramienta OpenSource que permite extraer métricas en general. Es un proyecto aprobado por la Cloud Native Computing Fundation

En NEuro se utiliza para monitorear las diversas métricas de Kubernetes: CPU, memoria, rendimiento de red, rendimiento de disco, etc.

4.6.12 GRAFANA

Grafana es una herramienta que nos permite visualizar datos desde multitud de fuentes.

Puede ser usada para monitorear métricas, logs o cualquier dato almacenado.

Uno de los puntos fuertes es que hay una base hay disponible una base de datos de muchos dashboards creados por la comunidad, y se pueden integrar simplemente sabiendo su ID.

4.7 GESTION DEL PROYECTO

4.7.1 AGILE Y SCRUM

Agile describe un conjunto de técnicas (o metodologias) para el desarrollo de proyectos en general (no solo software). Se trata de una definición de gestión del proyecto iterativa, en la que cada iteración se denomina Sprint.

La principal ventaja es que siguiendo metodologías Agile podemos tener un ciclo de vida adaptativo frente a cambios.

SCRUM, por su lado, define un conjunto de "buenas prácticas" y es un Framework de trabajo en equipo.

Contiene diferentes artefactos de desarrollo y diferentes roles para el equipo.

A menudo estos dos términos se confunden, Agile no es lo mismo que SCRUM. Agile no especifica los artefactos, roles, maneras de comunicación, etc. SCRUM si lo hace y es bastante restrictivo en ello.

El desarrollo de NEuro usa el Framework SCRUM junto con algunas recomendaciones de Agile.

4.7.2 JIRA DE ATLASSIAN

Se trata de una herramienta propiedad de la empresa Atlassian (de pago para equipos grandes y medianos) que ayuda con la gestión de proyectos SCRUM.

En NEuro ha sido la herramienta usada para la gestión del proyecto.

5 DESCRIPCIÓN DE LA SOLUCIÓN

En esta sección se hará una descripción simple pero completa del sistema, sin entrar en detalles técnicos específicos. Para conocer los detalles técnicos es necesario ir a los Anexos que se describen en la sección "Descripción de los anexos y entrega" de este mismo documento.

5.1 ARQUITECTURA

La arquitectura de NEuro ha sido diseñada para cumplir cuatro requisitos fundamentales:

- **Escalado funcional:** El sistema ha de soportar la adición de nuevos módulos sin modificar los existentes. Para satisfacer este requisito es fundamental tener un sistema de autenticación totalmente desacoplado de cualquier modulo
- **Escalado de rendimiento:** Dado que el numero potencial de usuarios es inmenso, es necesario dotar a NEuro de capacidades para soportar gran número de peticiones sin lastrar el rendimiento. Mas que rápido, ha de tener capacidad de soportar muchos usuarios.
- **Capacidad de recuperación:** cuando ocurra algo que bloquee alguno de los módulos ha de ser posible recuperarlos, idealmente de forma automática.
- **Inmutabilidad, trazabilidad y fiabilidad de los datos:** dado que hablamos de "dinero", es necesario guardar un histórico de los datos, además de una validación de estas múltiples veces. Para ello es ideal el uso de una blockchain, que además ha de ser privada y permisionada (Los datos son privados y además los usuarios han de estar autenticados para filtrar sus permisos)

Para cumplir con estos requisitos se ha desarrollado una arquitectura modular. El esquema de alto nivel puede entenderse con la siguiente imagen:

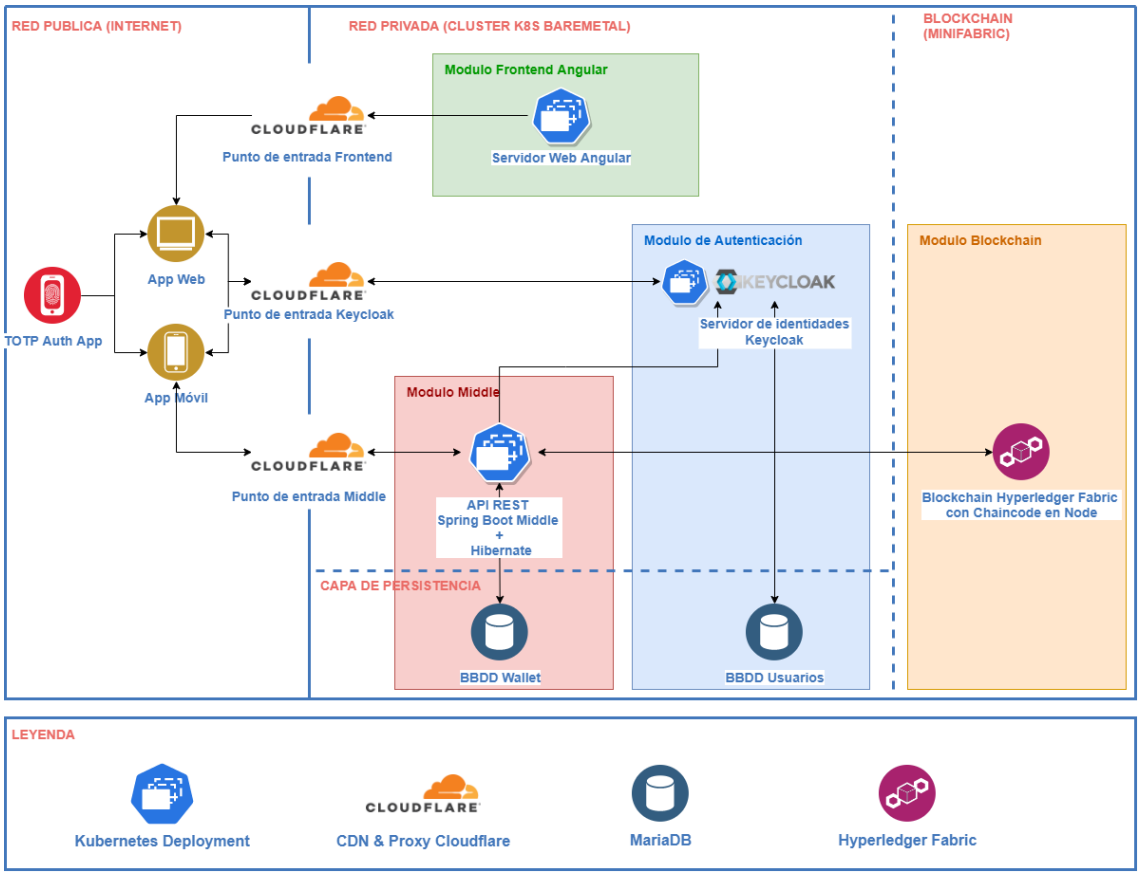


Ilustración 3: Arquitectura dividida por módulos

La comunicación entre los módulos se describe en el anexo "Arquitectura" al igual que la arquitectura a bajo nivel de cada módulo.

5.1.1 MÓDULO FRONTEND ANGULAR

Esta parte junto con el módulo de autenticación son las únicas con las que el usuario interactúa directamente, todo el resto de módulos están "tras bambalinas" y son totalmente transparentes para el usuario final.

Para tener una idea general es posible revisar el manual, disponible en anexo "Manuales". Además, en la sección "Acceso a los sistemas" de este mismo documento se explica cómo es posible acceder a la web real.

Dado que esta es la única parte con la que el usuario interactúa es de vital importancia que tenga una experiencia agradable y que se encuentre cómodo con el uso de la interfaz. Para ello NEuro permite que el usuario configure a su gusto los colores y efectos de la interfaz. Cuando un usuario configura algo a su gusto lo siente como suyo e indirectamente está mucho más cómodo usándolo.

Otro de los puntos clave aquí es informar en todo momento de las cosas que han ido bien (con mensajes en verde, por ejemplo) y de las que han ido mal (en color rojo, que siempre expresa que algo no ha ido bien o peligro). Así

pues, NEuro arroja un mensaje en verde cuando una operación ha terminado con éxito y en rojo cuando algo ha fallado, por ejemplo, una transferencia bancaria ha tenido un error en el proceso.

¿Como es posible que la web sepa quién soy?

Pues aquí entra el módulo de autenticación, que se va a encargar de establecer en el navegador del usuario todo lo necesario para su identificación, lo que vamos a llamar token de acceso JWT.

La autenticación requiere un nombre de usuario, una contraseña, y además un código de un solo uso generado en base al tiempo (obligatorio), y que solo sirve para un intervalo corto de tiempo. Esto añade un punto crucial de seguridad al sistema.

Con este token de acceso la web va a poder hacer llamadas al módulo middle.

5.1.2 MÓDULO MIDDLE

Este es el módulo central del sistema. Central no significa núcleo, simplemente significa que entre medias de todas las llamadas.

Cuando un usuario quiere hacer algo, el frontend iniciará una llamada, y esa llamada

¿Por qué es necesario un "middle" ?, podríamos hacer llamadas a la blockchcain y punto, ¿no?

Pues en realidad si, pero no es tan sencillo, para eso los desarrolladores del frontend (o de cualquier modulo nuevo que se integre) han de tener conocimientos sobre blockchain y conocimientos sobre certificados digitales, wallets, gRPC, Hyperledger Fabric y otras muchas tecnologías que no son nada sencillas.

Además de eso, el usuario tendría que estar también familiarizado con estos conceptos para poder hacer cosas tan sencillas como una transferencia ¿Quién sabe ahora mismo como pasar ethereum de una wallet fría a otra? Pues en realidad poca gente. En cambio ¿Quién sabe hacer una transferencia bancaria con la app del banco?

Visto esto el middle al final lo único que hace es pasar una llamada REST autenticada con JWT a una llamada Hyperledger Fabric autenticada con el sistema de Hyperledger (que se basa en certificados digitales).

El middle no contiene ninguna lógica, y ha de permanecer siempre desacoplado de cualquier lógica, simplemente se encargará de parsear llamadas.

5.1.3 MÓDULO BLOCKCHAIN

Este es el módulo clave de toda la arquitectura.

Una blockchain, en pocas palabras, y con poco rigor, no es más que una sucesión (cadena) de bloques que se almacena en muchos nodos. Estos bloques pueden entenderse como porciones de información de cualquier tipo, y en la blockchain siempre están ordenados.

Los nodos van a encargarse de ejecutar el código escrito para ellos (Chaincode o Smart Contracts) y el punto clave aquí es que todos ellos (Depende de la política de confirmaciones puede ser el 50% u otro, en NEuro podríamos bajar para ganar algo de rendimiento ya que los nodos son de confianza) han de obtener los mismos resultados de la ejecución de ese código.

Se dice que la ejecución tiene que ser determinista, es decir, que para unos mismos parámetros de entrada el resultado es siempre el mismo. Esto mantiene la información a salvo de nodos concretos, todos tendrían que ser comprometidos para poder alterar los datos.

En NEuro toda la lógica del negocio y las funcionalidades viven en este módulo, y así debe ser siempre.

Que una cuenta de usuario sea vulnerada pasa a ser menos peligroso, si un usuario reporta un ataque simplemente podríamos revisar el histórico de la cadena y revisar si es verdad, de esa manera podemos devolver su cuenta y revertir el ataque.

Por el hecho de usar una blockchain podemos olvidarnos del histórico de las transacciones y de los peligros que tiene una base de datos vulnerada. O quizá no olvidarnos completamente, pero si estar bastante más seguros que con una base de datos al uso.

Para detalles sobre como funciona la blockchain acudir al "Anexo II - Software" al apartado "2.4 FLUJO COMPLETO DE UNA CONSULTA, DESDE FRONT HASTA BLOCKCHAIN"

5.1.4 MODULO DE AUTENTICACIÓN

De toda la autenticación va a encargarse el servidor keycloak.

Al tener el módulo separado es posible hacer la autenticación de muchas maneras, no solo con el típico usuario y contraseña. En el caso de NEuro es obligatorio usar TOTP, solamente tenemos que instalar una APP en el móvil (por ejemplo, FreeOTP) y escanear el QR que se proporciona la primera vez que se inicia sesión.

5.1.5 OTROS ASPECTOS FUNDAMENTALES

Todo, excepto la blockchain, se ejecuta en contenedores Linux aislados, lo que permite utilizar soluciones de orquestación de contenedores.

Aquí entra en juego Kubernetes, que nos va a permitir escalar fácilmente cualquier parte del sistema (a excepción de la blockchain, que tiene su forma de escalar y es algo más compleja), poner más replicas o menos según convenga. Por ejemplo, podemos añadir el doble de replicas en fechas clave como en época de rebajas, donde normalmente se mueve mas dinero.

Cloudflare juega un papel fundamental también (o cualquier otro proxy/DNS). Exponer directamente los puntos de entrada de la infraestructura puede ser bastante peligroso, por eso se usa un proxy que hace como intermediario y que se encarga de ocultar nuestra IP real. Además, con Cloudflare podemos añadir protecciones contra ataques de denegación de servicio u otros ataques típicos.

NEuro tiene una integración y entrega continua, que permite despliegues rápidos (cruciales a la hora de corregir problemas críticos). Con subir el código nuevo al repositorio GitLab ya se van a lanzar todos les test automáticos, y si estos tienen éxito, se desplegará directamente en producción. Con un "click" hemos actualizado todo el sistema, incluida la blockchain y además con el sistema siempre operativo (gracias a Kubernetes)

6 ACCESO A LOS SISTEMAS

Todos los accesos a los sistemas serán eliminados en febrero de 2022, tanto los usuarios como las entradas DNS y accesos desde Internet.

6.1 NEURO WEB

- URL: neuro-front.felipesanchezweb.es

Dado que la web tiene varios roles, a continuación, se enumeran los usuarios por rol que se han creado para las muestras.

Las Imágenes TOTP son códigos QR que han de escanearse con la app "Free TOTP" de Red Hat.

Hay que tener la hora del móvil correctamente configurada. La hora exacta, incluidos segundos. Es posible que debido a desfases temporales entre el servidor y el móvil haya algún momento en el que TOTP no funcione como se espera, en un entorno productivo real esto no ocurre dado que la hora se sincroniza con errores de milisegundos, en mis servidores la hora no es tan

crítica y no se sincroniza tan a menudo, o incluso puede haber conexiones lentas, problemas con los servidores de tiempo entre otros que hagan que no funcione del todo bien.

La web no está optimizada para ser utilizada desde un móvil, es necesario acceder a ella desde un ordenador.

Administrador de Identidades:

- Usuario: **admin-docs**
- Contraseña: **+Q3zVEy6?L**
- Imagen TOTP:

Hacienda pública:

- Usuario: **hacienda-docs**
- Contraseña: **+Q3zVEy6?L**
- Imagen TOTP

Ciudadano regular:

- Usuario: **12341234A**
- Contraseña: **+Q3zVEy6?L**
- Imagen TOTP

Empresa:

- Usuario: **Z12341234ES**
- Contraseña: **+Q3zVEy6?L**
- Imagen TOTP

6.2 RUTA DE PRUEBA DE UN PROCESO COMPLETO (EJEMPLO)

A modo de ayuda, a continuación se define una posible ruta de prueba paso a paso para.

Esta ruta es solo un ejemplo, para mas información consultar el anexo con los manuales.

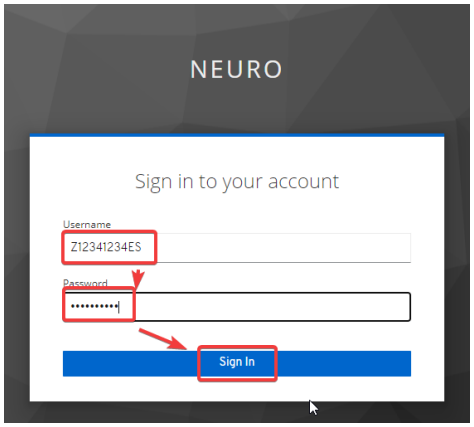
Recuerdo que el sistema dejará de ser accesible una vez obtenida la calificación.

Escaneo de códigos TOTP

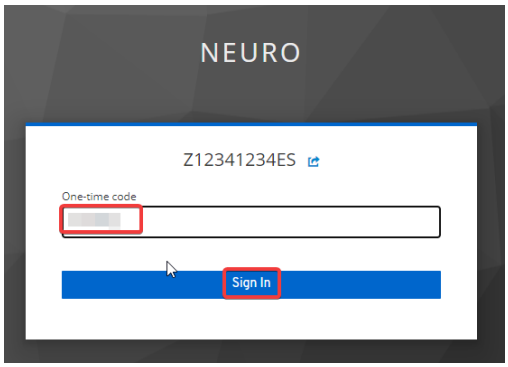
1. Instalar la App "FreeOTP" (Desarrollada por Redhat) disponible en la tienda de aplicaciones del smartphone.
2. Con ella, escanear (pulsando sobre el icono del código QR) los códigos QR del "ciudadano regular" y de "la empresa" y la "hacienda pública", disponibles un poco más arriba en este mismo documento.

Emisión de factura por la empresa

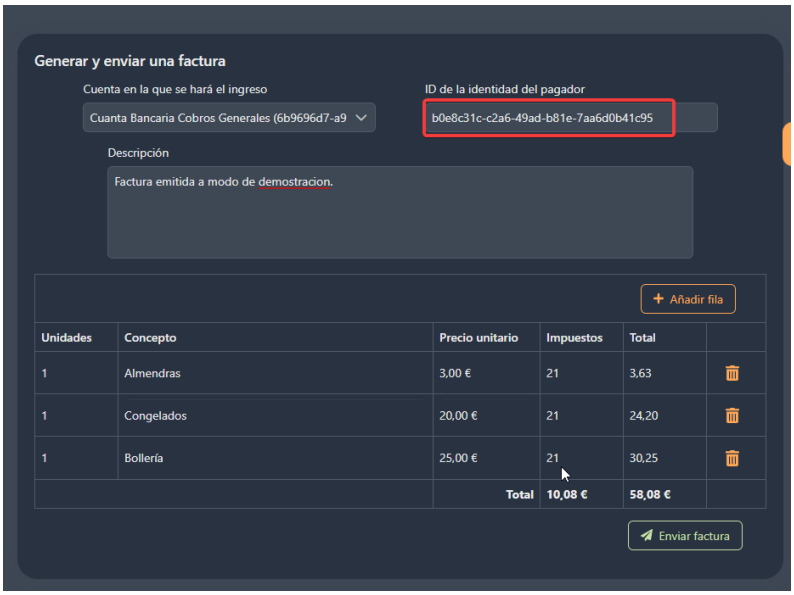
3. A continuación, acceder a neuro-front.felipesanchezweb.es , introducir el nombre de usuario de la empresa (**Z12341234ES**) y la contraseña (**+Q3zVEy6?L**).



4. Abrir la aplicación FreeOTP y obtener el código de un solo uso que se da desde la misma.



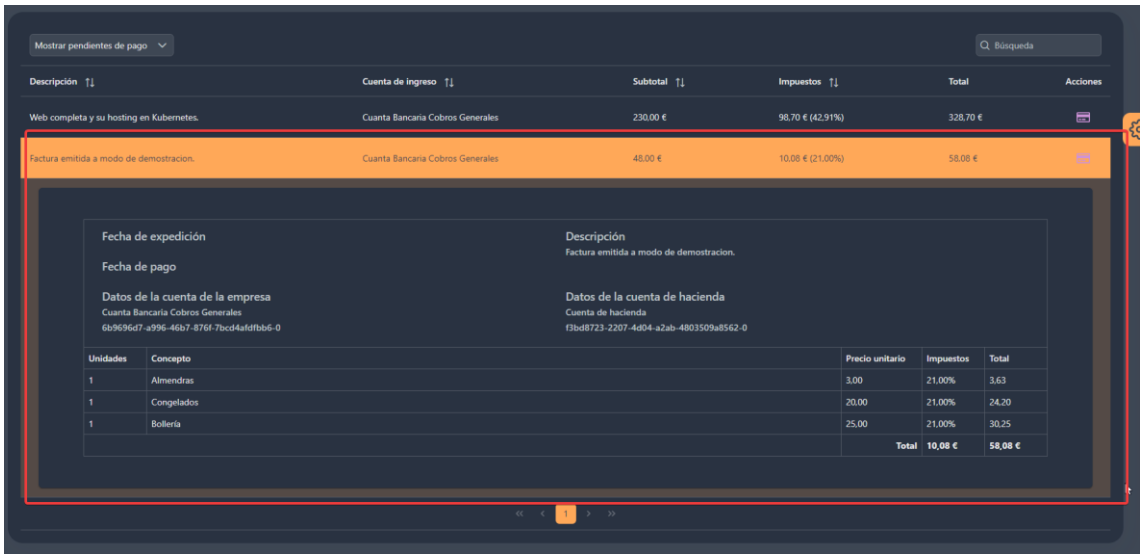
5. Una vez dentro del sistema vamos a emitir una factura. Desde el menú acceder a "Facturas", "Emitir factura". Rellenar los campos con el "ID de la identidad del pagador" siguiente: **b0e8c31c-c2a6-49ad-b81e-7aa6d0b41c95-0** (Corresponde al ID del ciudadano). Terminar de rellenar la factura como se quiera.



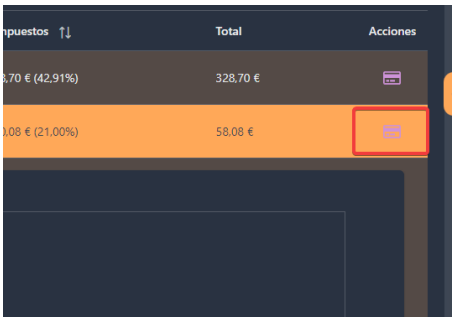
6. Enviar factura. Esto tomará unos segundos (del orden de 20-30 segundos). Esto generará una transacción que almacenará la factura en la blockchain para siempre.

Revisión y pago de la factura por parte del ciudadano.

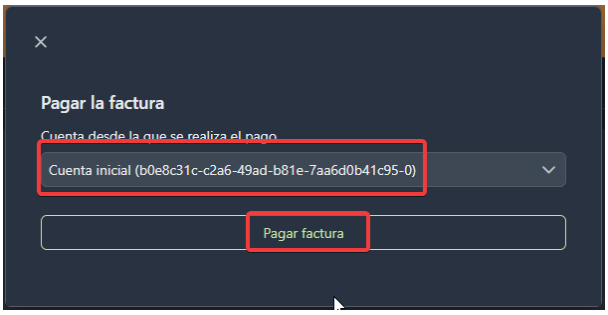
7. Acceder a neuro-front.felipesanchezweb.es , introducir el nombre de usuario del ciudadano (**12341234A**) y la contraseña (**+Q3zVEy6?L**). De manera análoga al punto 4, introducir el código de un solo uso y acceder al sistema.
8. Acceder a las facturas pendientes desde "Facturas", "Mis facturas pendientes". Ha de aparecer la factura emitida anteriormente.

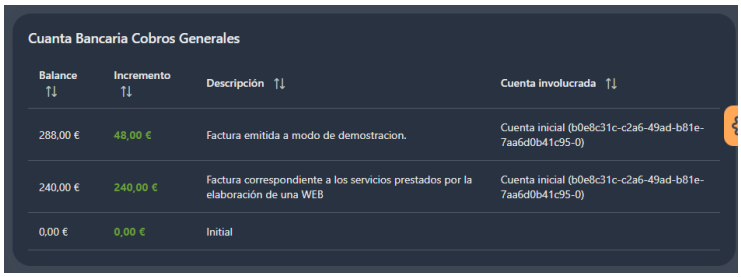


9. Para pagarla simplemente hacer click en el icono de la tarjeta de crédito.



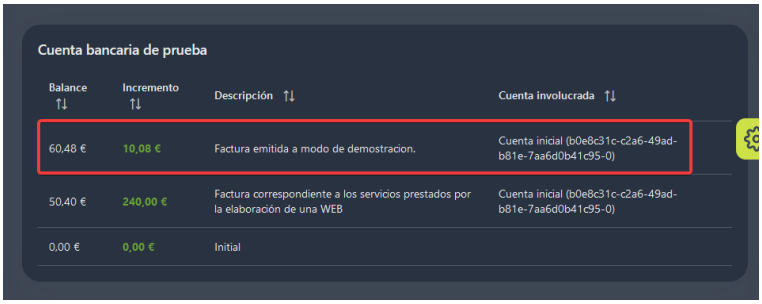
10. Seleccionar la cuenta desde la que se hará el pago y hacer click en "Pagar factura". Después de unos segundos la factura será pagada. El importe sin impuestos irá a la cuenta de la empresa, y los impuestos a la cuenta de la hacienda.





Revisión de los impuestos pagados en la cuenta de la hacienda

11. Acceder a neuro-front.felipesanchezweb.es , introducir el nombre de usuario de la hacienda pública (**hacienda-docs**) y la contraseña (**+Q3zVEy6?L**). De manera análoga al punto 4, introducir el código de un solo uso y acceder al sistema.
12. Acceder a "Mis cuentas bancarias". Después acceder al historial haciendo click en el icono del reloj y revisar como se han transferido los impuestos



6.3 MONITOREO CON GRAFANA

- URL**:** **grafana.felipesanchezweb.es**
- Usuario: **viwer@viwer.viwer**
- Contraseña: **+Q3zVEy6?L**

7 REFERENCIAS

[1] Javascript
   a. https://developer.mozilla.org/es/docs/Web/JavaScript
   b. https://es.wikipedia.org/wiki/JavaScript
[2] Typescript
   a. https://www.typescriptlang.org/
   b. https://openwebinars.net/blog/que-es-typescript/
   c. https://www.typescriptlang.org/docs/handbook/basic-types.html#tuple
[3] NodeJS
   a. https://nodejs.org/es/
[4] Hyperledger Fabric
   a. https://www.ibm.com/es-es/topics/hyperledger
   b. https://academy.bit2me.com/cuantos-tipos-de-blockchain-hay/
   c. https://www.hyperledger.org/use/fabric
   d. https://hyperledger-fabric.readthedocs.io/en/release-1.4/arch-deep-dive.html
[5] MySQL:
   a. https://es.wikipedia.org/wiki/MySQL
[6] Spring Boot / Spring Framework
   a. https://www.campusmvp.es/recursos/post/que-son-spring-framework-y-spring-boot-tu-primer-programa-java-con-este-framework.aspx
[7] Angular
   a. https://medium.com/somoswigou/angular-vs-react-vs-vue-cu%C3%A1l-es-la-mejor-opci%C3%B3n-941a207951c7
[8] PrimeNG
   a. https://www.primefaces.org/layouts/freya-ng
   b. https://www.primefaces.org/primeng/
[9] Keycloak
   a. https://www.keycloak.org/
[10] OpenID
   a. https://es.wikipedia.org/wiki/OpenID
[11] TOTP
   a. https://www.onelogin.com/learn/otp-totp-hotp?_bt=260722100289&_bk=&_bm=b&_bn=g&utm_source=GOOGLE&utm_medium=cpc&gclid=CjwKCAjw47eFBhA9EiwAy8kzNK-KnA2KQfC3VQnBODaB7TyhID8Pgw9VrcCYVz9tv7VKigXx9K89JxoCvxQQAvD_BwE
   b. https://en.wikipedia.org/wiki/Time-based_One-Time_Password
[12] SSO
   a. https://es.wikipedia.org/wiki/Single_Sign-On

[13] Proxmox:
a. https://www.qemu.org/
b. https://pve.proxmox.com/wiki/Main_Page
c. https://www.proxmox.com/en/
d. https://es.wikipedia.org/wiki/Kernel-based_Virtual_Machine
e. https://es.wikipedia.org/wiki/LXC
f. https://xcp-ng.org/
g.
[14] Debian:
a. https://ubunlog.com/debian-y-ubuntu-ganadores-en-el-mercado-de-servidores-linux/#:~:text=Con%20un%20resultado%20similar%2C%20Ubuntu,por%20encima%20de%20las%20siguientes.
b. https://es.wikipedia.org/wiki/Debian
c. https://www.debian.org/index.es.html
[15] GitLab:
a. https://es.wikipedia.org/wiki/GitLab
b. https://www.redhat.com/es/topics/devops/what-is-ci-cd
[16] PfSense
a. https://www.pfsense.org/products/
b. https://www.pfsense.org/
c. https://www.pfsense.org/getting-started/
d. http://www.haproxy.org/
e. https://docs.netgate.com/pfsense/en/latest/packages/acme/index.html
f.
[17] Minifabric
a. https://github.com/hyperledger-labs/minifabric
b. https://es.wikipedia.org/wiki/Ansible_(software)
[18] Kubernetes
a. https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/
b. https://kubernetes.io/es/
[19] MetalLB
a. https://metallb.universe.tf/
[20] Lens
a. https://k8slens.dev/
[21] Portainer
a. https://www.portainer.io/
[22] Cloudflare
a. https://www.cloudflare.com/es-es/
b. https://es.wikipedia.org/wiki/Cloudflare
[23] Agile y SCRUM
a. https://www.grupocibernos.com/blog/agile-vs-scrum-cual-eliges

b. https://universidadalnus.com/diferencia-entre-agile-y-scrum/#:~:text=Agile%20describe%20un%20conjunto%20de,practica%20el%20desarrollo%20de%20software.
[24] Pushover
a. Fhttps://pushover.net/
[25] Prometheus
a. https://prometheus.io/

ANEXO I - ARQUITECTURA

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En el presente anexo se describirá detalladamente cada módulo de la arquitectura del sistema (nada que ver con la arquitectura de software).

En cada módulo se especificará su forma de comunicación con otros módulos y los argumentos de la elección de esa forma de comunicación.

2 MÓDULOS

El sistema se compone de tres módulos y principales y el módulo de autenticación. En la siguiente imagen puede verse cada módulo bien definido en cuatro colores.

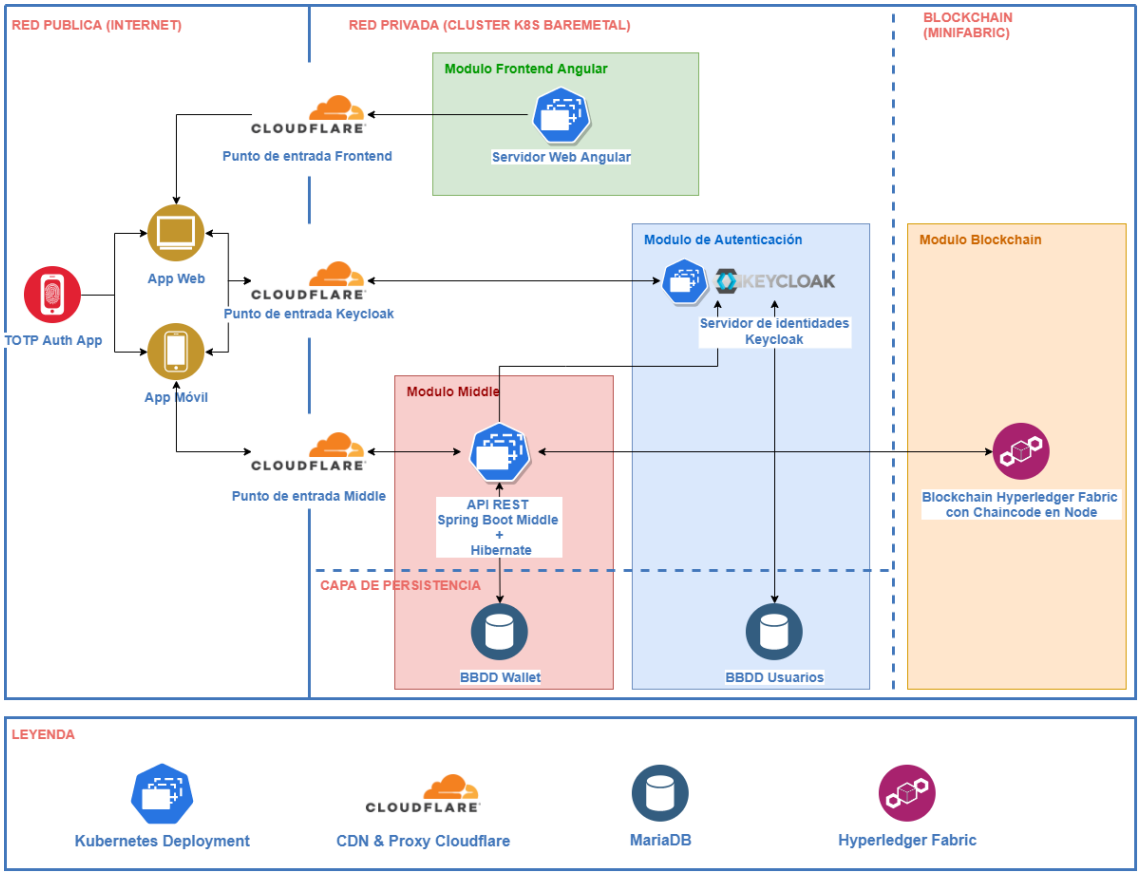


Ilustración 1: Arquitectura por módulos del sistema NEuro

Los módulos que se ven en la imagen:

- **Módulo Frontend Angular:** aquí se encuentra contenida la aplicación web Angular
- Módulo Middle: este módulo maneja todas las comunicaciones con la blockchain.
- Módulo Blockchain: este módulo es la propia blockchain en si misma
- Módulo de Autenticación: se trata del servidor de autenticaciones.

2.1 MÓDULO FRONTEND ANGULAR

Este módulo contiene la aplicación Angular.

Es el módulo final de toda la cadena y es con el único que un usuario interactúa directamente.

Este módulo está completamente preparado para añadir todos los idiomas necesarios, aunque actualmente solo existe el Español. Para ello, todos y cada uno de los literales han sido extraídos a un fichero de traducción.

2.1.1 COMUNICACIONES

Este módulo ejecuta sus comunicaciones desde el navegador del cliente dado que es ahí donde se ejecuta.

Las comunicaciones son:

- Con el módulo Middle
- Con el módulo de Autenticación (Keycloak)
- Con un posible servidor de logs

2.1.1.1 COMUNICACIÓN CON EL MÓDULO MIDDLE

Esta comunicación es la más importante dado que es la que soporta toda la lógica de negocio.

Se hace mediante REST contra la API que expone el Middle.

Toda llamada al módulo Middle ha de llevar en sus cabeceras un token JWT con la identificación del usuario que hace la llamada.

A continuación, se pueden ver todos los datos que se almacenan en el session storage del navegador y que vienen del módulo de autenticación:

Key	Value
refresh_token	eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJiODI2...
nonce	TDhuNIBDdlBaTzdTbTV3aTRoUXpzNVoySDF6WUo4MGV0...
id_token_expires_at	1623011407000
expires_at	1623011407205
access_token_stored_at	1622579407205
id_token_stored_at	1622579407206
id_token_claims_obj	{"exp":1623011407,"iat":1622579407,"auth_time":1622579...
id_token	eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhdkV...
access_token	eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhdkV...
PKCE_verifier	cFphdU1iU3JtZUVpei5IM04uSm1LMEJkQ2FJflR4eHJzSVdt...
session_state	e6c44a0d-7f92-4d90-beef-b4501ad96a05
granted_scopes	["openid email offline_access profile"]

Ilustración 2: Ejemplo de información almacenada por Keycloak en el "Session Storage" del navegador

A destacar:

- access_token: JWT de autenticación con el Middle
- refresh_token: Token para refrescar el acces_token

2.1.1.2 COMUNICACIÓN CON EL MÓDULO DE AUTENTICACIÓN

A su vez, la aplicación Angular se comunica mediante el protocolo OpenID con el módulo de autenticación.

Esta comunicación da como resultado los tokens JWT que luego se utilizan para identificarse con el Middle.

2.1.1.3 COMUNICACIÓN CON UN SERVIDOR DE LOGS

En la aplicación se ha añadido además NgxLoggerLevel para dar soporte al almacenamiento externo de logs.

En las aplicaciones que se ejecutan del lado del cliente en un navegador no disponemos directamente de los logs de los usuarios, por esta razón lo que se hace es almacenarlos en algún servidor mediante llamadas de algún tipo.

Aunque la aplicación está preparada para ello, no se ha implementado ningún servidor que guarde esos logs, y las llamadas han sido bloqueadas.

Con NgxLoggerLevel es muy sencillo añadir esta funcionalidad. Basta con configurarlo de la siguiente manera:

```
LoggerModule.forRoot({level: NgxLoggerLevel.WARN, serverLogLevel:
NgxLoggerLevel.ERROR}),
```

Esta línea lo que hace es simplemente es decirle a Angular que importe el Módulo LoggerModule de NgxLoggerLevel y lo configura con niveles de

advertencia y error para los logs en en navehador y los log en el servidor externo respectivamente.

2.2 MÓDULO MIDDLE

El módulo Middle puede considerarse el núcleo del sistema en cuanto a comunicaciones.

No es el núcleo lógico ya que no tiene ninguna lógica de negocio, simplemente enmascara llamadas y las prepara para la blockchain.

En este módulo se encuentra la base de datos mas critica y la que mas hay que proteger: la base de datos de las Wallets ("BBDD Wallet"). Las Wallets en realidad son los certificados digitales expedidos por la CA de Hyperledger Fabric. estos certificados autentican a los usuarios en la blockchain, por lo que si alguien malicioso se hiciera con ellos podría operar en nombre de cualquier usuario.

A continuación, se puede ver una imagen en la que se presenta la arquitectura interna en cuanto a comunicaciones:

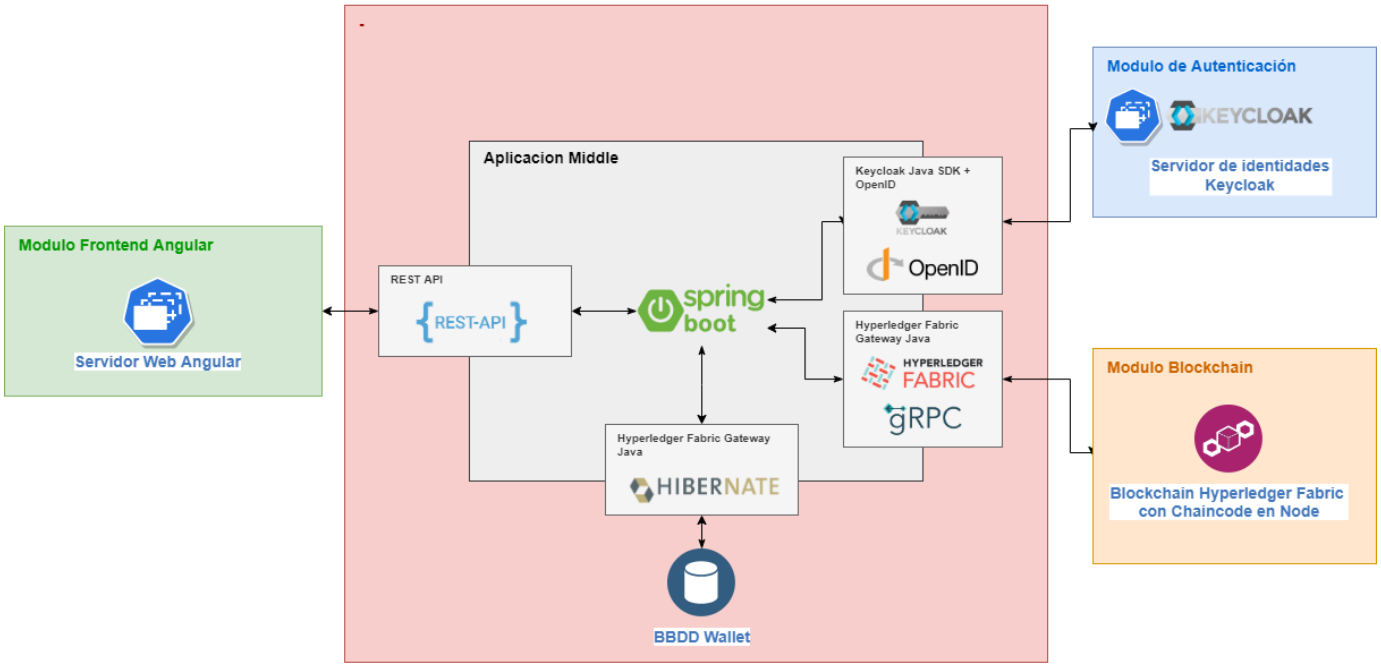


Ilustración 3: Arquitectura interna por componentes del Middle del sistema NEuro

2.2.1 COMUNICACIONES

2.2.1.1 COMUNICACIÓN CON EL MÓDULO FRONTEND ANGULAR

Las comunicaciones con el Módulo Frontend Angular las empieza siempre el Frontend y las realiza a través de una API REST.

El módulo Middle simplemente expone esos endpoints para ser llamados.

2.2.1.2 COMUNICACIÓN CON EL MÓDULO DE AUTENTICACIÓN

El Middle simplemente se comunica con el Módulo de Autenticaciones para:

- Verificar la autenticación y sus roles
- Crear nuevos usuarios o identidades

Todas las llamadas que el Frontend hace requieren de roles específicos y autenticación dependiendo de la acción que quiera realizar.

2.2.1.3 COMUNICACIÓN CON EL MÓDULO BLOCKCHAIN

Aquí es donde se establece una de las comunicaciones más importantes del sistema.

Como ya he dicho, la función del Middle es simplemente enmascarar las llamadas a la Blockckchain, estas llamadas se hacen a través de la biblioteca Fabric Gateway Java que proporciona funcionalidad para conectarse a la blockchain e invocar al Chaincode.

Dado que este paso tiene cierta complejidad en código, se explicará en el "Anexo II -Software" como es un flujo de una llamada desde el endpoint REST hasta la blockchain.

2.3 MÓDULO BLOCKCHAIN

Este es el módulo más complejo y corresponde a toda la blockchain.

Aunque en NEuro para su primera versión (la entregada en este TFG) este módulo contiene lo básico de Hyperledger Fabric, a medida que el sistema se fuera adoptando seria necesario añadir nuevos nodos.

Para entenderlo bien hay que tener primero en cuenta la siguiente imagen, que presenta cada uno de los nodos de los diferentes tipos y que ejecuta en su interior:

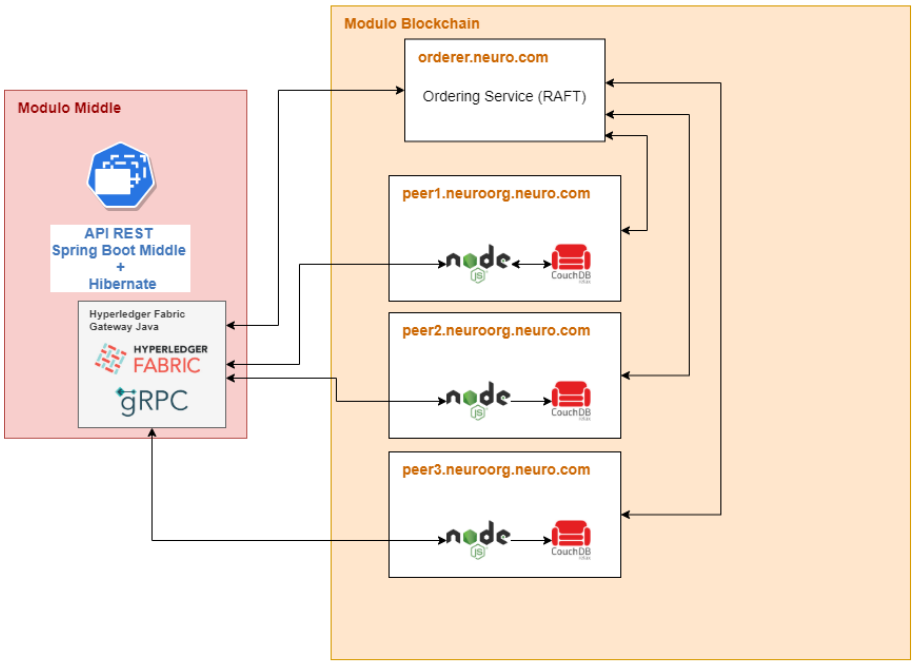


Ilustración 4: Arquitectura de los nodos y módulos de la blockchain (resumido) del sistema NEuro.

Por el momento no voy a entrar en detalle sobre el funcionamiento de HL Fabric ya que eso se explicará posteriormente en el anexo "Anexo II - Software".

No obstante, es posible entender a la perfección como funciona el flujo de las transacciones en HL Fabric desde la documentación oficial: Documentación HL Fabric.

Lo que si que quiero es explicar es que significa "DRAFT": se trata de la implementación recomendada para el servicio de ordenación, otra opción es usar Kafka, pero esta declarada como obsoleta. El servicio de pedidos o de ordenación (Ordering Service) se encarga de tramitar las peticiones con las transacciones, ordenarlas (para que el libro mayor o cadena de bloques siempre tenga un estado coherente) y, en caso de que todo sea correcto, avisar de que se añadirá el nuevo bloque con las nuevas transacciones a la blockchain.

2.3.1 COMUNICACIONES

En cuento a comunicaciones externas aquí no hay mucho que comentar, simplemente se comunica con el módulo Middle a través del SDK para Java.

Nota: HL Fabric contiene el SDK para lo clientes y el SDK para los Smart Contracts. Siempre que se hable de Java y el SDK se hace referencia al cliente, no al de los Smart Contracts

2.3.1.1 COMUNICACIÓN CON EL MÓDULO MIDDLE

Esta comunicación se hace usando el SDK de Hyperledger Fabric para Java que internamente utiliza gRPC (gRPC es un framework de llamadas a procedimientos remotos, Remote Procedure Call, de alto rendimiento).

Estas llamadas van autenticadas mediante el certificado digital que se expide desde el nodo CA para cada usuario.

3 REFERENCIAS

[1] gRPC
   a. https://grpc.io/

ANEXO II – SOFTWARE Y ESTRUCTURA DE DATOS

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En el presente documento se describe el funcionamiento del software en su totalidad, detallando los aspectos más relevantes de cada módulo.

Como ya viene siendo habitual, se procederá a explicar individualmente el frontend Angular, el Middle, y el Chaincode de la blockchain.

Para finalizar el apartado de software se describirá también como los diferentes módulos interactúan a nivel de código y se pondrá un ejemplo de flujo completo desde el frontend hasta la blockchain.

Por otro lado, se describe la estructura de datos, que como ya he mencionado anteriormente es no relacional.

2 SOFTWARE

2.1 BLOCKCHAIN

Bien, para explicar el código usado en la blockchain, primero hay que entender cómo funciona el código en Hyperledger Fabric.

Actualmente, (mediados de 2021) HL Fabric ofrece de manera oficial SDKs (Software Developmen Kit, un conjunto de herramientas listas para ayudar en el desarrollo) para Go, Java y NodeJS. Para neuro se ha decidido usar NodeJS por muchos motivos, pero los mas importantes son los siguientes:

- Angular está hecho con Typescript, por lo que las interfaces y clases del chaincode se pueden compartir con el frontend.
- Es un lenguaje que interactúa directamente con JSON, por lo que usar JSON es trivial.
- Typescript funciona con Javascript y por tanto tiene una de las comunidades de desarrolladores más grandes que existen.

Sabiendo esto, se puede entender un poco mejor cómo funciona el SDK de HL Fabric.

Para crear un contrato inteligente solo es necesario heredar de la clase Contract que proporciona el SDK, cualquier clase que herede de Contract ya es un contrato inteligente:

```
export class CheckingAccountContract extends Contract {
}
```

Uno o más contratos inteligentes pueden empaquetarse en el mismo chaincode.

Para especificar los contratos inteligentes que contiene el chaincode se hace de la siguiente manera:

```
export const contracts: any[] = [
    NEuroIdentityContract,
    CheckingAccountContract,
    InvoiceSystemContract,
    TaxAuthoritiesContract
];
```

Dicho con palabras: se meten todas las clases en una lista llamada "contracts" y se exporta a todos los módulos.

Aunque no es necesario dividir la aplicación en varios contratos, es muy útil para estructurar el código. Además, es perfectamente factible instanciar un contrato dentro de otro y utilizar sus funciones.

Ahora que ya sabemos como crear contratos inteligentes vacíos, es hora de añadir los métodos, o las "acciones", que en lenguaje de blockchain se denominan transacciones.

Para crear una transacción basta con utilizar la anotación "@Transaction()" del SDK. Además, también podemos especificar transacciones que solo leen, en lenguaje de la blockchain diremos que estas transacciones no ejecutan confirmación o "commit" en la blockchain, por lo tanto, no escriben nada.

```
@Transaction()
public async createCheckingAccount(ctx: Context, displayName: string): Promise<CheckingAccount> {
    //TODO: hacer algo
}

@Transaction(false)
public async readMyCheckingAccount(ctx: Context, accountId: string): Promise<CheckingAccount> {
    //TODO: Hacer algo que de solo lectura
}
```

Por norma general las operaciones de solo lectura son mucho más rápidas ya que no requieren confirmación por todos los nodos. Podemos asumir que en un determinado momento una app bancaria "funcione mal" y no muestre la cantidad correcta a cambio de velocidad, pero no podemos asumir de ninguna manera que esa cantidad incorrecta se use para hacer operaciones, por ejemplo, transferencias.

Hay que tener en cuenta que las transacciones han de tener el mismo resultado se ejecuten en el par que sea y en el tiempo que sea. Por ejemplo, si tenemos una operación que guarda en base de datos la hora actual en milisegundos, DARA PROBLEMAS DE INCONSISTENCIA dado que la transacción no se ejecuta en el mismo milisegundo.

¿Qué ocurre si hay inconsistencias y dos pares obtienen resultados distintos en alguna operación?: Pues sencillamente la transacción se revertirá, y no se confirmará, nada más, nada se va a romper. Algunas operaciones que suelen ser propensas a transacciones fallidas:

- Obtención de fechas y horas
- Acceso a API externas
- Generación de números aleatorios

Esto es tan importante que lenguajes como Solidity (Para Smart Contracts en Ethereum) tienen limitaciones funcionales, y ni siquiera permiten hacer estas operaciones. Personalmente prefiero tener yo la responsabilidad y que se me permita decidir qué hacer y que no hacer, asumiendo transacciones fallidas.

Bien, ya entendemos cómo se hacen los contratos inteligentes, a continuación, se va a presentar el código de uno de los que utiliza NEuro, concretamente el que opera con las cuentas bancarias, que puede considerarse uno de los más importantes:

```
@Info({title: 'CheckingAccountContract', description: 'This smart contract contains functionality needed to ope
rate with a CheckingAccount.' })
export class CheckingAccountContract extends Contract {

    @Transaction(false)
    @Returns('boolean')
    public async checkingAccountExists(ctx: Context, accountId: string): Promise<boolean> {
        const buffer = await ctx.stub.getState(accountId);
        return (!!buffer && buffer.length > 0);
    }

    @Transaction(false)
    @Returns('boolean')
    public async checkingAccountIsMine(ctx: Context, accountId: string): Promise<boolean> {
        const identityContract = new NEuroIdentityContract();
        const identity = await identityContract.readMyIdentity(ctx);
        if (identity.checkingAccounts.filter(account => account === accountId).length === 1) {
            return true;
        }else{
            return false;
        }
    }


    @Transaction()
    public async createCheckingAccount(ctx: Context, displayName: string): Promise<CheckingAccount> {
        const identityId = ContextUtils.getIdentityUtilIdFromContext(ctx);
        const identityContract = new NEuroIdentityContract();
        const identity = await identityContract.readMyIdentity(ctx);
        const newCheckingAccountIndex = identity.checkingAccounts.length;
```

```
        const compositeKey =  ctx.stub.createCompositeKey(CheckingAccount.name, [identityId, newCheckingAccount
Index.toString()]);
        if (await this.checkingAccountExists(ctx, compositeKey)) {
            throw new Error(`The CheckingAccount ${compositeKey} already exists`);
        }
        const newAccount = new CheckingAccount();
        newAccount.id = compositeKey;
        newAccount.balance = 0;
        newAccount.displayName = displayName;
        const lastMovement = new LastMovementData();
        lastMovement.description = 'Initial';
        lastMovement.increment = 0;
        newAccount.lastMovement = lastMovement;

        await ctx.stub.putState(compositeKey, Buffer.from(JSON.stringify(newAccount)));

        identity.checkingAccounts.push(compositeKey);
        const identityDBKey = ctx.stub.createCompositeKey(NEuroIdentity.name, [identityId]);
        await ctx.stub.putState(identityDBKey, Buffer.from(JSON.stringify(identity)));

        return newAccount;
    }


    @Transaction(false)
    @Returns('CheckingAccount')
    public async readMyCheckingAccount(ctx: Context, accountId: string): Promise<CheckingAccount> {

        if (!await this.checkingAccountIsMine(ctx, accountId)) {
            throw new Error(`The CheckingAccount ${accountId} not exists in your Identity`);
        }

        return await this.readCheckingAccount(ctx, accountId);
    }

    @Transaction(false)
    @Returns('CheckingAccount')
    public async listCheckingAccounts(ctx: Context): Promise<CheckingAccount[]> {
        const identityContract = new NEuroIdentityContract();
        const identity = await identityContract.readMyIdentity(ctx);
        const accountList: CheckingAccount[] = [];

        for (const accountId of identity.checkingAccounts){
            const account = await this.readCheckingAccount(ctx, accountId);
            accountList.push(account);
        }

        return accountList;
```

```
    }

    @Transaction()
    @Returns('CheckingAccount')
    public async transferAmmount(ctx: Context, fromAccountId: string, toAccountId: string, amount: number, desc
ription: string): Promise<CheckingAccount> {

        if (!await this.checkingAccountIsMine(ctx, fromAccountId)) {
            throw new Error(`The CheckingAccount ${fromAccountId} not exists in your Identity`);
        }

        if (!await this.checkingAccountExists(ctx, toAccountId)) {
            throw new Error(`The CheckingAccount ${toAccountId} not exists`);
        }

        if ( fromAccountId === toAccountId ) {
            throw new Error(`The CheckingAccount from and to cant be the same`);
        }

        const fromAccount: CheckingAccount = await this.readCheckingAccount(ctx, fromAccountId);

        if (fromAccount.balance < amount) {
            throw new Error(`The CheckingAccount ${toAccountId} does not have enough amount to transfer ${amoun
t} NEuro`);
        }

        const toAccount: CheckingAccount = await this.readCheckingAccount(ctx, toAccountId);

        // WARNING!
        // Amount transfer
        fromAccount.balance -= amount;
        const fromLastMovement = new LastMovementData();
        fromLastMovement.increment = - amount;
        fromLastMovement.involvedCheckingAccount = CheckingAccount.reduceToMiniCheckingAccount(toAccount);
        fromLastMovement.description = description;
        fromAccount.lastMovement = fromLastMovement;

        toAccount.balance += amount;
        const toLastMovement = new LastMovementData();
        toLastMovement.increment = + amount;
        toLastMovement.involvedCheckingAccount = CheckingAccount.reduceToMiniCheckingAccount(fromAccount);
        toLastMovement.description = description;
        toAccount.lastMovement = toLastMovement;

        await this.updateCheckingAccount(ctx, fromAccountId, fromAccount);
        await this.updateCheckingAccount(ctx, toAccountId, toAccount);

        return fromAccount;
    }
```

```
    /**
     * Get the history of the account balances
     */
    @Transaction(false)
    @Returns('HistoryRow[]')
    public async readCheckingAccountHistory(ctx: Context, accountId: string): Promise<HistoryRow[]> {
        const historyIterator: Iterators.HistoryQueryIterator = await ctx.stub.getHistoryForKey(accountId);
        const allResults: HistoryRow[] = [];
        while (true){
            const res = await historyIterator.next();
            if (res.value) {
                const current = res.value as any;
                current.value = JSON.parse(res.value.value.toString('utf8')) as CheckingAccount;
                allResults.push(current as HistoryRow);
            }

            if (res.done) {
                await historyIterator.close();
                return allResults.reverse();
            }
        }
    }

    // TODO: Eliminar este metodo
    @Transaction()
    @Returns('CheckingAccount')
    public async printBills(ctx: Context, accountId: string, amount: number, description: string): Promise<Chec
kingAccount> {
        if (!await this.checkingAccountIsMine(ctx, accountId)) {
            throw new Error(`The CheckingAccount ${accountId} not exists in your Identity`);
        }
        if (amount <= 0){
            throw new Error(`Cant print ${amount}, it must be >= 0`);
        }

        const account = await this.readCheckingAccount(ctx, accountId);

        account.balance += amount;
        const lastMovement = new LastMovementData();
        lastMovement.increment = + amount;
        lastMovement.description = 'Prinded Bills 😁 😁 😁';
        account.lastMovement = lastMovement;

        await this.updateCheckingAccount(ctx, accountId, account);

        return account;
```

```
    }


    public async readCheckingAccount(ctx: Context, accountId: string): Promise<CheckingAccount> {
        const buffer = await ctx.stub.getState(accountId);
        return JSON.parse(buffer.toString()) as CheckingAccount;
    }

    public async updateCheckingAccount(ctx: Context, accountId: string, newCheckingAccount: CheckingAccount): P
romise<void> {
        const buffer = Buffer.from(JSON.stringify(newCheckingAccount));
        await ctx.stub.putState(accountId, buffer);
    }

}
```

Aunque el fragmento de código parezca muy grande, no son más que 200 líneas de código.

Dado que no tiene sentido explicar todo el código, simplemente voy a extraer una porción del código anterior, y voy a añadir comentarios a cada operación para que se pueda ver detalladamente (incluso con demasiada información) que se está haciendo y como. En particular se trata de la operación de creación de cuentas bancarias:

```
    // Este metodo crea una nueva cuenta bancaria asociada a la identidad que hace la invocacion.
    @Transaction()
    public async createCheckingAccount(ctx: Context, displayName: string): Promise<CheckingAccount> {
        // Se obtiene el ID de la identidad de quien ejecuta la transaccion del contexto de la transaccion
        // En particular se obtiene del certificado digital asociado que invoca la transaccion.
        const identityId = ContextUtils.getIdentityUtilIdFromContext(ctx);
        // Se instancia el contrato inteligente de las Identidades
        const identityContract = new NEuroIdentityContract();
        // Se lee la identidad de quien ejecuta la transaccion
        const identity = await identityContract.readMyIdentity(ctx);
        const newCheckingAccountIndex = identity.checkingAccounts.length;

        // Se genera una clave compuesta para identificar la cuenta bancaria.
        // Las claves compuestas de HL Fabric no son mas que multiples string separados por el caracter nulo, oj
o, nulo no
        // es lo mismo que vacio.
        const compositeKey =  ctx.stub.createCompositeKey(CheckingAccount.name, [identityId, newCheckingAccountI
ndex.toString()]);
        if (await this.checkingAccountExists(ctx, compositeKey)) {
            throw new Error(`The CheckingAccount ${compositeKey} already exists`);
        }
        // Se crea la cuenta añadiendo los datos necesarios, balance, descripciones de ultimos movimientos, etc.
        const newAccount = new CheckingAccount();
        newAccount.id = compositeKey;
```

```
        newAccount.balance = 0;
        newAccount.displayName = displayName;
        const lastMovement = new LastMovementData();
        lastMovement.description = 'Initial';
        lastMovement.increment = 0;
        newAccount.lastMovement = lastMovement;

        // Se empuja la cuenta a la base de datos (ledger o libro mayor)
        await ctx.stub.putState(compositeKey, Buffer.from(JSON.stringify(newAccount)));

        // Se añade la cuenta a las cuentas del usuario (o mas correctamente de la identidad)
        identity.checkingAccounts.push(compositeKey);
        const identityDBKey = ctx.stub.createCompositeKey(NEuroIdentity.name, [identityId]);
        // Se empuja la modificacion de la identidad
        await ctx.stub.putState(identityDBKey, Buffer.from(JSON.stringify(identity)));

        return newAccount;
    }
```

Con el funcionamiento de esta transacción ya es posible hacerse una idea de cómo son el resto.

Como se puede ver las operaciones no son muy distintas a las que se harían con cualquier base de datos no relacional. En concreto las operaciones son muy similares a las que se harían directamente usando CoachDB

2.2 MIDDLE

El middle utiliza Spring Boot y por ello la arquitectura básica es a base de Controllers y Services.

Voy a empezar explicando las partes sencillas del middle y posteriormente explicaré la creación de usuarios, que es algo compleja dado que se crean en Keycloak y en la Blockchain.

Antes que nada, hay que recordar que ahora estamos con Java en vez de Typescript, por lo que, aunque muy similar, la sintaxis cambia un poco.

2.2.1 OPERACIONES BÁSICAS GENÉRICAS

2.2.1.1 CONTROLLERS Y SERVICES

De manera análoga a como se creaban los Smart Contracts, un controller se crea con la anotación @RestController (En este caso se trata de un controller REST, es decir, que admite endpoints REST):

```
@RestController
@RequestMapping("chaincode/checkingaccounts")
@RolesAllowed({RolesConstants.ROLE_USER,
RolesConstants.ROLE_BUSINESS_USER, RolesConstants.ROLE_TAX_AUTHORITY})
```

```
public class CheckingAccountController {
}
```

Aquí hay varias cosas por explicar:

- **@RestController**: anotación que se usa para indicar que la clase es un controlador de SpringBoot.
- **@RequestMapping("chaincode/checkingaccounts")**: Anotación para indicar que el controller admite llamadas REST en las rutas "**chaincode/checkingaccounts**". Siempre definiendo los endpoints de estas llamadas dentro del controller.
- **@RolesAllowed({RolesConstants.ROLE_USER, RolesConstants.ROLE_BUSINESS_USER, RolesConstants.ROLE_TAX_AUTHORITY})**: Esta anotación indica los roles que tienen acceso al controller. Todos los endpoints definidos dentro van a tener esa regla de acceso. Esta anotación esta configurada para que trabaje a nivel de los roles keycloack del cliente definido para el middle.

Todos los controllers van a ser declarados de la misma manera.

Una vez que emos entendido los controllers, vamos a pasar a los servicios. Los servicios son usados para separar la lógica de negocio en una capa diferente a los controladores. Se crean simplemente añadiendo la anotación "@Service":

```
@Service
public class NEuroContract {
}
```

Los servicios se instancian en los controladores con la anotación @Autowired, que indica al controlador que tiene que hacer la instanciación de ese servicio:

```
@RestController
@RequestMapping("chaincode/checkingaccounts")
@RolesAllowed({RolesConstants.ROLE_USER,
RolesConstants.ROLE_BUSINESS_USER, RolesConstants.ROLE_TAX_AUTHORITY})
public class CheckingAccountController {
    @Autowired
    NEuroContract contract;
}
```

De igual forma se puede instanciar un servicio dentro de otro si es necesario.

2.2.1.2 LÓGICA DE LOS ENDPOINTS

2.2.1.3 CONCEPTOS PREVIOS DE LA LÓGICA DE LOS ENDPOINTS

Primero vamos a entender algunos conceptos que posteriormente se nombrarán.

2.2.1.3.1 WALLET SQL (HLFABRICSQLWALLET)

EL concepto de Wallet en HL Fabric hace referencia a una lista que almacena pares clave-valor.

Las claves son los ID de las identidades, en Neuro estos ID son los UUID que se generan desde Keycloak al crear usuarios.

Los valores son, de forma simplificada, "certificados digitales" que contienen las identidades de los usuarios.

En el caso de NEuro, se ha creado una Wallet customizada que almacena esta lista en una base de datos. Para crear una Wallet customizada basta con implementar la interfaz "Wallet" del sdk de HL Fabric (paquete org.hyperledger.fabric.gateway.Wallet).

2.2.1.4 LA LÓGICA

Dado que este módulo no es más que un wrapper que transforma las llamadas REST en llamadas a la blockchain, se ha desarrollado una forma genérica aplicable a cualquier llamada.

Lo primero que hay que hacer es definir el cuerpo de la llamada REST. Esto se hace Creando una clase Java la cual ha de tener definidos los atributos junto con las anotaciones de validación necesarias (estas anotaciones vienen de javax.validation.constraints):

```
package com.neuro.middle.app.models.endpointsbody.checkingaccounts;

import lombok.Data;
import javax.validation.constraints.NotNull;

@Data
public class NewCheckingAccountBody {
    @NotNull
    private String displayName;
}
```

La anotación @Data viene de lombok y no es mas que un acortador o shortcut para indicar al editor que genere Getters, Setters y contructores genericos.

Esta forma de definir los cuerpos de llamadas es la primera, y dado que obligaba a hacer ciertas operaciones repetidas a la hora de matear las propiedades/atributos del cuerpo con los argumentos de las transacciones de HL Fabric, se elaboró una anotación Java para facilitar esa tarea:

```
package com.neuro.middle.app.anotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
```

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface ChaincodeFieldOrder {
    //This must start in 0!
    int argPos() default 1;
}
```

Esta anotación lo que permite es definir la posición de las propiedades en los argumentos de las transacciones, se utiliza así:

```java
public class PayInvoiceBody {

    @ChaincodeFieldOrder(argPos = 0)
    private String invoiceId;

    @ChaincodeFieldOrder(argPos = 1)
    private String checkingAccountId;

}
```

Esto va a indicar posteriormente que "invoiceId" es el argumento en posición 0, y "checkingAccountId" en posición 1.

Bien, una vez definido el cuerpo o body de la solicitud (en llamadas GET por ejemplo no será necesario por no tener cuerpo), podemos seguir definiendo el endpoint.

Siguiendo con el ejemplo de PayInvoice, el endpoint se crearía así:

```java
package com.neuro.middle.app.controllers.fabric;

import com.neuro.middle.app.models.endpointsbody.invoicesystem.PayInvoiceBody;
import com.neuro.middle.app.models.endpointsbody.invoicesystem.RegisterNewInvoiceBody;
import com.neuro.middle.app.services.hlfabric.chaincode.NEuroContract;
import com.neuro.middle.app.utils.constants.RolesConstants;
import com.neuro.middle.app.utils.functions.FunctionsUtils;
import org.hyperledger.fabric.gateway.ContractException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.annotation.security.RolesAllowed;
import java.beans.IntrospectionException;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.security.Principal;
import java.util.concurrent.TimeoutException;

@RestController
@RequestMapping("chaincode/invoices")
public class InvoiceSystemController {

    @Autowired
    NEuroContract contract;

    private final String CONTRACT_NAME = "InvoiceSystemContract";

    private final String METHOD_REGISTER_NEW_INVOICE = "issueInvoice";
    private final String METHOD_LIST_ISSUED_INVOICES = "listIssuedInvoices";
```

```
    private final String METHOD_LIST_INVOICES_TO_PAY = "listInvoicesToPay";
    private final String METHOD_PAY_INVOICE = "payInvoice";


    @RolesAllowed({RolesConstants.ROLE_USER, RolesConstants.ROLE_BUSINESS_USER})
    @PostMapping("pay")
    public ResponseEntity<Object> payInvoice(Principal principal, @RequestBody
PayInvoiceBody body) throws IOException, InterruptedException, TimeoutException,
ContractException, IllegalAccessException, IntrospectionException,
InvocationTargetException {
        Object r =  this.contract.submitTransactionWithClassAsArgs(principal,
                this.CONTRACT_NAME, this.METHOD_PAY_INVOICE,
                body, body.getClass(), Object.class);
        return new ResponseEntity<>(r, HttpStatus.OK);
    }
}
```

Como se ve en el código, para definir la clase a la que se mapea el cuerpo se define como argumento en el método del endpoint:

```
@RequestBody PayInvoiceBody body
```

Que en el método quedaría así:

```
ResponseEntity<Object> payInvoice(Principal principal, @RequestBody PayInvoiceBody body)
```

Uno de los puntos clave aquí es la llamada a "this.contract" (Servicio):

```
Object r =  this.contract.submitTransactionWithClassAsArgs(principal,
        this.CONTRACT_NAME, this.METHOD_PAY_INVOICE,
        body, body.getClass(), Object.class);
```

Esta llamada se encarga de ejecutar la transacción del método definido con los parámetros del body en el orden especificado por la anotación "@ChaincodeFieldOrder" que ya vimos antes como se usa.

Aquí es donde esta toda la magia de la forma desarrollada para hacer llamadas genéricas.

No obstante, para entenderlo aun mejor, vamos a ver como funciona el servicio según el código:

```
public <T> T submitTransactionWithClassAsArgs(Principal principal,
                                              String contractName,
                                              String functionName,
                                              Object inputObject,
                                              Class<?> inputType,
                                              Class<T> outputType) throws IOException,
IllegalAccessException, IntrospectionException, InvocationTargetException,
InterruptedException, TimeoutException, ContractException {
    Transaction transaction =  this.createTransaction(
            FunctionsUtils.getSubjectWFromPrincipal(principal),
            contractName,
            functionName
    );

    String[] props = FunctionsUtils.getChaincodePropsInOrder(inputObject, inputType);
     return FunctionsUtils.objectFromJSONBytes(transaction.submit(props), outputType);
}
```

Aquí la lógica empieza a ponerse compleja y lo mejor es acudir al código que se entrega para entenderlo. No obstante, voy a intentar explicar que está sucediendo.

Primero se crea una transacción utilizando una función auxiliar:

```
public Transaction createTransaction(String identityId, String contractName ,String
contractMethod) throws IOException {

    Gateway.Builder builder = Gateway.createBuilder();
    builder.identity(wallet, identityId).networkConfig(new
FileInputStream(hlFabricConfigService.getConfigFile())).discovery(false);
    Gateway gateway = builder.connect();

    // get the network and contract
    Network network = gateway.getNetwork(channelName);
    Contract contract = network.getContract(this.contractName, contractName);
    return contract.createTransaction(contractMethod);
}
```

Para crear la transacción lo que se hace es obtener la identidad de la wallet SQL y se genera una nueva instancia de la red, y con ella se obtiene una instancia del Smart Contract especificado. Una vez tenemos la instancia del Smart contract, solo queda generar la transacción necesaria indicando el método del Smart Contract que se va a ejecutar.

Lo único que queda es preparar los parámetros de la transacción. Para ello se hace uso de la función de utilería "getChaincodePropsInOrder" que inteligentemente va a retornar una con las propiedades del body ordenadas.

Cuando ya tenemos la transacción generada y los parámetros en orden, solo queda enviar esa transacción para que sea ejecutada por los peers.

2.3 FRONTEND

En cuanto al frontend no voy a comentar partes del código de NEuro ya que cada componente es totalmente diferente del resto.

Como ya he mencionado, el frontend está hecho en su totalidad en Angular. Este framework tiene una estructura/arquitectura muy definida y que hay que seguir. Angular además, y a diferencia de React o Vue contiene todo lo necesario para realizar una aplicación si necesidad de recurrir a terceros.

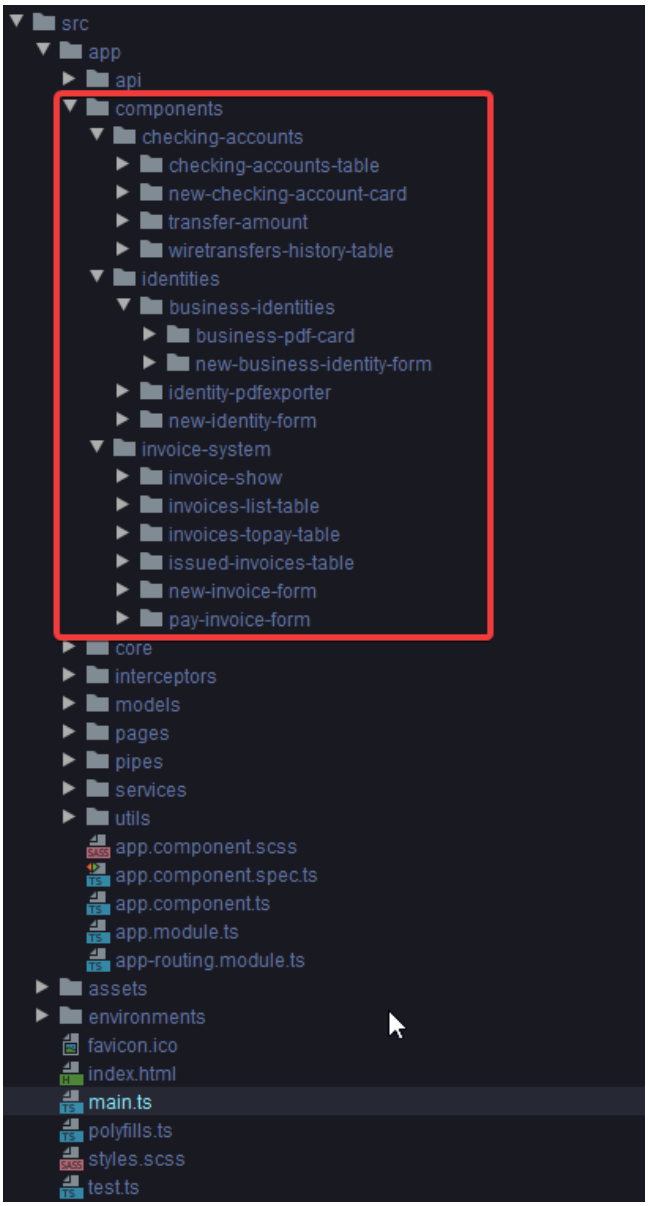
Toda la aplicación es multi idioma, y aunque actualmente solo se soporta el castellano, esta preparada para admitir cualquier idioma. Para ello se hace uso de la biblioteca "ngx-translate", que permite tener archivos de traducciones separados de los componentes. Estos archivos se definen en archivos JSON, que son cargados como assets, después simplemente se usan en los componentes de esta manera:

```
<div class="p-field p-fluid">
    <label for="toAccountId">{{ "checking-accounts.wiretransfers.labels.to" | translate }}</label>
    <span>
        <input id="toAccountId" type="text" pInputText class="ng-invalid ng-dirty" formControlName="toAcco
               [ngClass]="{'ng-invalid ng-dirty': formSent && fControls.toAccountId.errors}">
```

Otro punto clave destacable de la app Angular es el uso de "ngx-logger", el cual permite mandar a un servidor los logs de la aplicación. Esto puede ser importante en un futuro para identificar puntos de falla habituales de la aplicación mientras se ejecuta en un navegador. De otra manera los logs morirían en el navegador sin llegar a mandarse a ninguna parte.

Al estar usando un tema de pago de PrimeNG, lo que ellos proporcionan es un proyecto con todo el scaffolding (andamiaje o estructura del codigo) ya predefinido y con varias páginas de ejemplo en las que se puede ver cómo usar los diferentes componentes de PrimeNG.

La estructura de los componentes del proyecto es auto explicativa:



Los componentes se encuentran bajo el directorio "components" y se van distribuyendo en otros subdirectorios según su funcionalidad.

La biblioteca "angular-oauth2-oidc" permite usar Keycloak para gestionar la autenticación.

2.4 FLUJO COMPLETO DE UNA CONSULTA, DE FRONT A BLOCKCHAIN

Dado que es un poco complejo entender como funciona, a continuacion se presenta un diagrama de secuencia en el que se puede ver como se van distribullendo las llamadas:

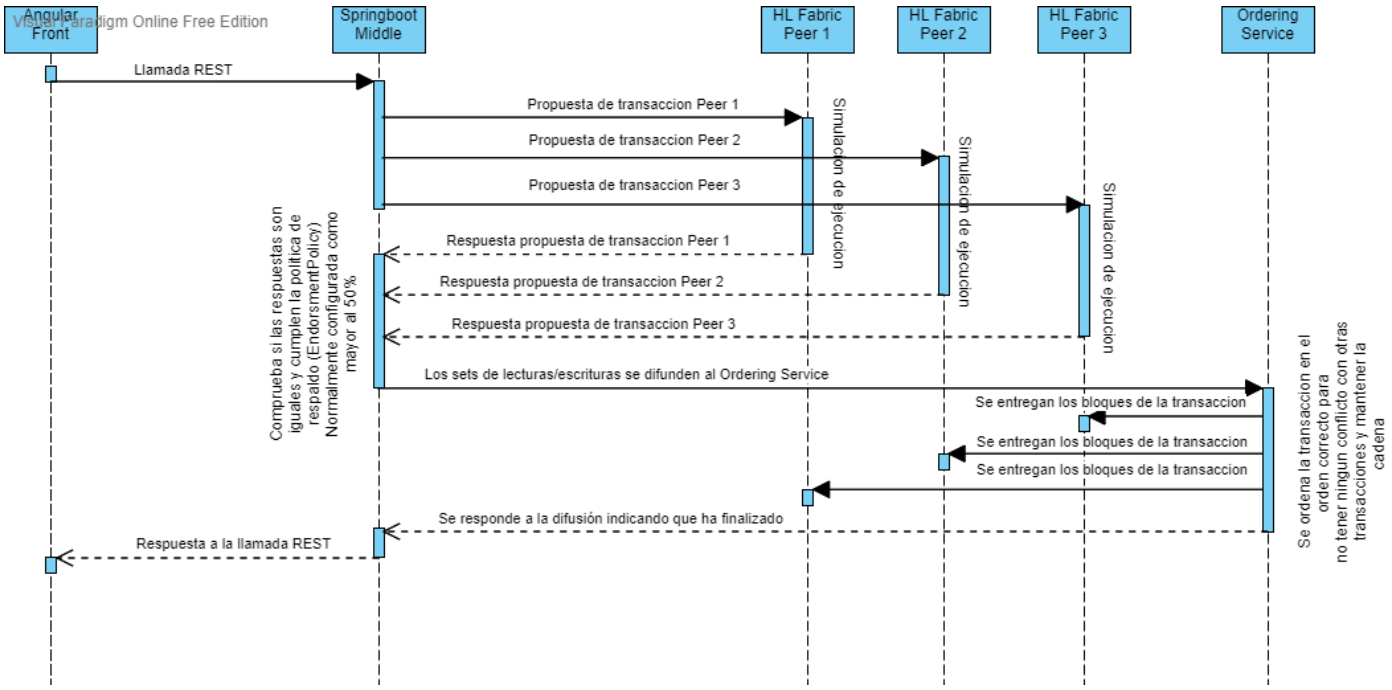


Ilustración 1: Diagrama de secuencia con una llamada completa desde el frontend a la blockchain

Podemos decir que hay cuatro pasos detrás de la blockchain, y dos mas si añadimos la llamada/respuesta REST. A continuación se van a describir todos los pasos, incluidas las llamadas REST (Pasos 1 y 6):

1. **Inicio del proceso, llamada REST al middle:** este primer paso hace referencia simplemente a la llamada rest que se hace desde Angular al Middle Springboot
2. **Envio de propuesta de transacción a los Peers y su ejecución:** los Peers de la blockchain reciben la propuesta que el SDK de HL Fabric manda desde el Middle Springboot, y la ejecutan para obtener una respuesta. Esta respuesta es vital dado que de ella depende si la transacción es acepta a (Si se cumple la política de endorsment) o no.
3. **Respuesta de la ejecución de la propuesta de transacción por los peers:** la respuesta de los Peers se retorna al SDK y este hace una evaluación previa para verificar la política de ensorsment. En el caso

de transacciones de solo lectura aquí termina el proceso y simplemente se hace la respuesta de la llamada REST (Paso 6).

4. **envió de los conjuntos de lecturas/escrituras de las anteriores propuestas de transacción al servicio de ordenación y ordenación de la transacción:** una vez aceptada la verificación previa por el SDK, se envían los conjuntos de lecturas/escrituras (Y la información de la transacción) al servicio de ordenación. Este servicio verifica el orden de la transacción, de modo que las lecturas y escrituras se hagan en el orden correcto y manteniendo la integridad del libro mayor (Ledger o base de datos de la blockchain).
5. **Respuesta del servicio de ordenación:** si el servicio de ordenación ordena correctamente la transacción y decide que es hora de añadirla a la blockchain, simplemente se avisará a los Peers, al SDK y al perr de confirmaciones (Este peer no se ha añadido en el esquema por simplicidad). Ahora ya tenemos el nuevo bloque añadido a la cadena, y en el la información modificada por la transacción.
6. **Respuesta de la llamada REST del paso 1:** finalmente el middle retorna la llamada REST a Angular.

Todo esto he descrito toma menos de 10 segundos. Es por ello por lo que HL Fabric se hace ver como una blockchain empresarial. Una transacción similar en Ethereum por ejemplo tomaría más de medio minuto en el mejor de los casos, y dependiendo de lo que paguemos por ella, si pagamos poco por el gas se puede demorar incluso varios minutos…

3 ESTRUCTURA DE DATOS

La estructura de los datos viene dada por los modelos typescript ya que se almacenan de forma no relacional en CouchDB. En otras palabras, se guarda un JSON (que en realidad será la clase completa sin funciones) con los datos correspondientes.

EL diagrama de clases es el siguiente (Generado con la extensión para VS Code "classdiagram-ts"):

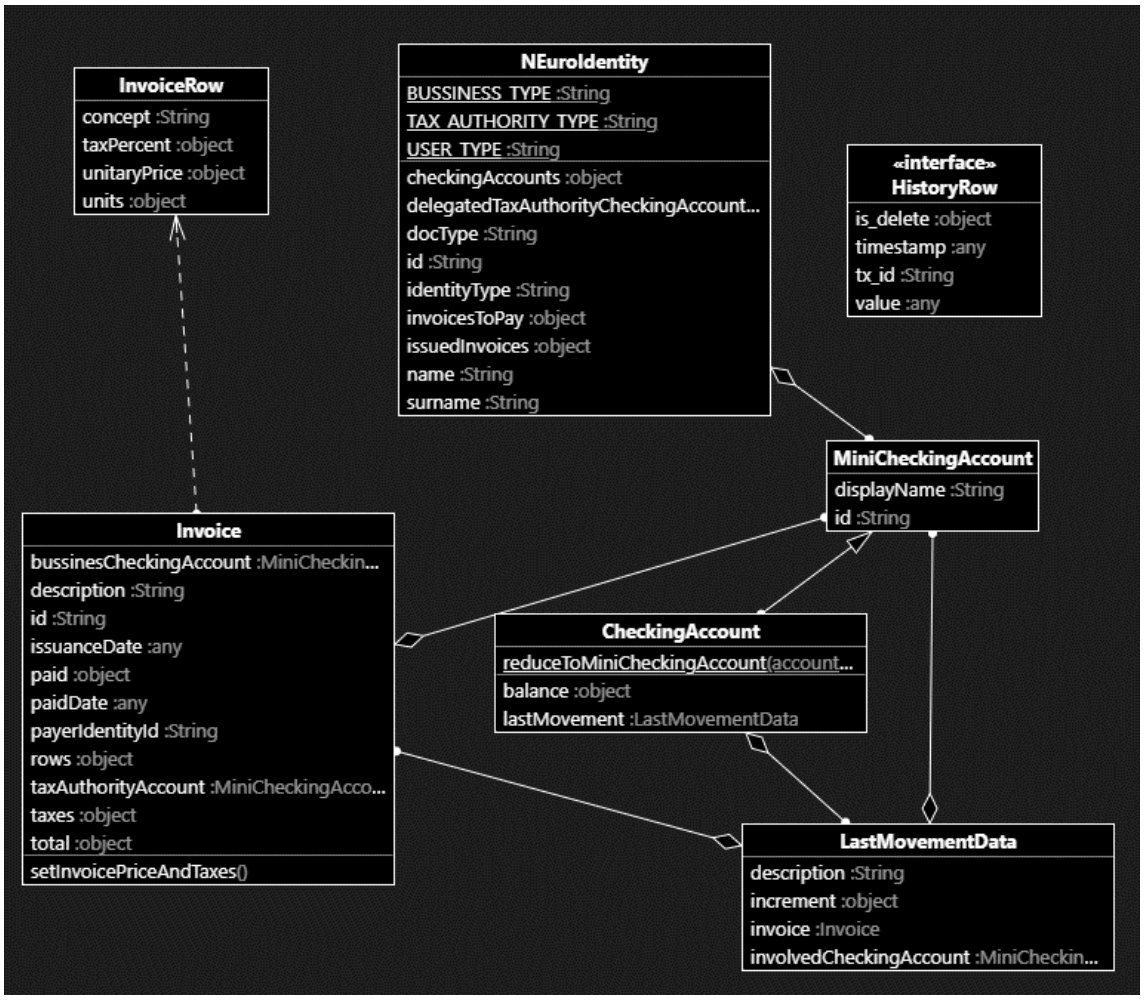


Ilustración 2: Diagrama de clases de los datos almacenados en la blockchain

4 REFERENCIAS

https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode4ade.html

https://projectlombok.org/

https://www.primefaces.org/primeng/

https://www.primefaces.org/freya-ng/#/

https://github.com/manfredsteyer/angular-oauth2-oidc

https://github.com/dbfannin/ngx-logger

https://github.com/ngx-translate/core

ANEXO III - INFRAESTRUCTURA

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En el presente anexo se va a definir toda la infraestructura utilizada para el despliegue en el entorno de demostración, que es equivalente a producción.

Además, como punto adicional se definirá un entorno blockchain ideal.

2 INFRAESTRUCTURA DESPLEGADA

2.1 MAQUINA FÍSICA

La infraestructura desplegada para NEuro desplegada completamente en una maquina HP Proliant con:

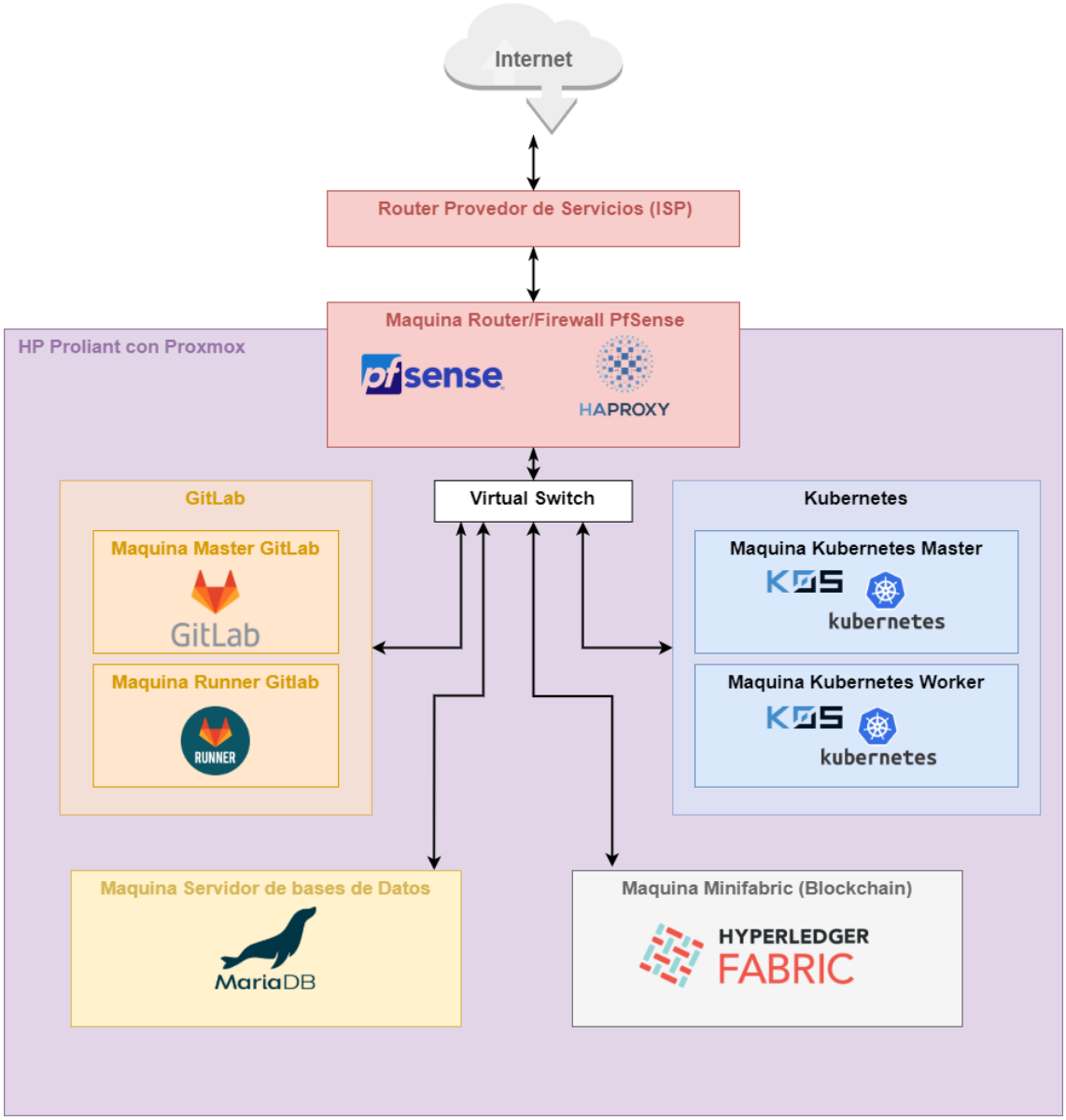
- 2 Intel Xeon de 10 núcleos 20 hilos, entre los dos 40 hilos
- 70 GB RAM
- 3TB de almacenamiento SSD en RAID 5, 2TB al final
- 1 TB HHD sin RAID

Esta máquina física ejecuta el sistema operativo Debian, y encima de el, se ejecuta Proxmox para gestionar las máquinas virtuales

2.2 MAQUINAS VIRTUALES

Dado que utilizar un servidor físico por maquina seria extremadamente caro y poco flexible, se ha decidido virtualizar todo el entorno, incluidas las redes.

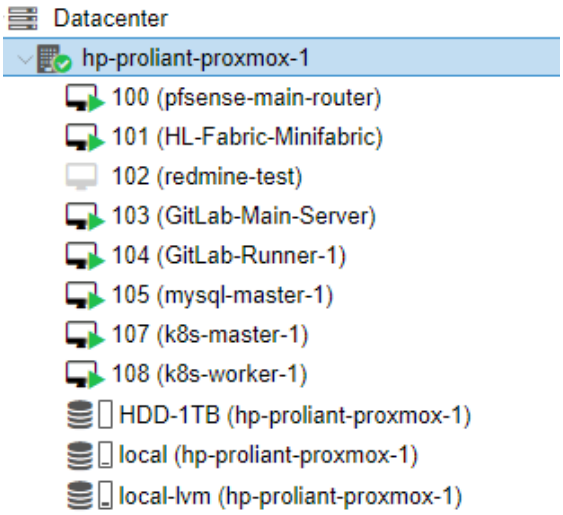
Las máquinas virtuales que se ejecutan en Proxmox y las que contienen todo el sistema son las siguientes:



En total son siete maquinas, que montarlas físicas seria extremadamente caro, tanto por consumo energético como por recursos.

Mantener estas máquinas en el HP Proliant supone un gasto mensual estimado de 11€. Unos 105 vatios hora.

A continuacion una muestra de todas las maquinas que se encuentran en Proxmox:



En las siguientes secciones se va a ir detallando los recursos de cada máquina. Estos recursos han sido seleccionados a partir de diferentes pruebas y se han ido ajustando para obtener los necesarios sin que sobren muchos.

2.2.1 ROUTER/FIREWALL PFSENSE

Esta máquina es una de las principales dado que es el router de la red interna.

En PfSense además se han configurado varios servicios más:

Services Status

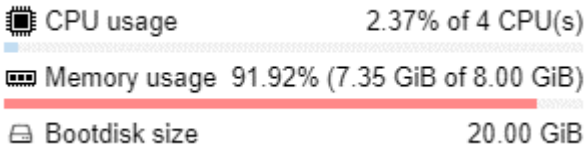
Service	Description	Action
dhcpd	DHCP Service	
dpinger	Gateway Monitoring Daemon	
haproxy	TCP/HTTP(S) Load Balancer	
iperf	iperf Network Performance Testing Daemon/Client	
ntpd	NTP clock sync	
sshd	Secure Shell Daemon	
syslogd	System Logger Daemon	
unbound	DNS Resolver	

A destacar:

- **HAProxy**: que actúa como proxy inverso y balanceador de carga
- **Unbound**: que funciona como DNS interno privado, además, redirecciona las peticiones a otros DNS externos si no encuentra la dirección solicitada

- **DHCP Server:** servidor DHCP, que además tiene configuradas varias IP que asigna de manera estática asociadas a direcciones MAC especificas.

Recursos y consumo real:



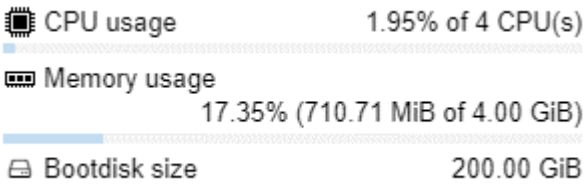
Se puede ver un consumo muy alto de memoria, esto es por que HAProxy esta configurado para usar toda la memoria que pueda como caché.

2.2.2 SERVIDOR DE BASE DE DATOS

Estea maquina ejecuta Debian con un servidor de bases de datos MariaDB (Version Open Source de MySQL)

Se incluyen las bases de datos del Middle y de Keycloak

Recursos y consumo real:



2.2.3 MASTER KUBERNETES

La maquina ejecuta Debian, y mediante k0s se ejecuta el nodo maestro de Kubernetes.

Una forma simple de crear ese maestro es la siguiente:

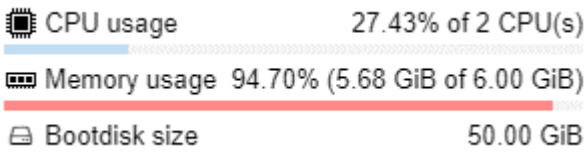
```
wget https://github.com/k0sproject/k0s/releases/download/v0.10.0/k0s-
v0.10.0-amd64 -O /usr/local/bin/k0s
chmod +x /usr/local/bin/k0s
# Configuracion por defecto, para crear una configuracion usar:
# k0s default-config > k0s-config.yml
# k0s install server -c k0s-config.yml
k0s install server
systemctl daemon-reload
systemctl enable k0sserver
systemctl start k0sserver


# C R E A C I O N   D E   T O K E N S
```

```
# k0s token create --role=worker --expiry="100h"
```

Primero se descarga el binario de k0s. Después se instancia el nodo maestro (server k0s) y se habilita.

La última línea corresponde a la creación del token que se usará para la creación de los nodos workers

Recursos y consumo real:



2.2.4 WORKER KUBERNETES

Como sistema operativo se usa Debian, y por encima se ejecuta un nodo worker de Kubernetes generado por K0s

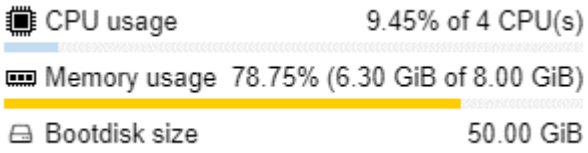
De manera similar al nodo maestro, se puede generar un worker de esta manera:

```
#Configurar el token generado desde el master
workerToken='...'

curl -sSLf k0s.sh | sh
k0s install worker
sed -
i "s#ExecStart=/usr/local/bin/k0s worker#ExecStart=/usr/local/bin/k0s wor
ker $workerToken#g" /etc/systemd/system/k0sworker.service
systemctl daemon-reload
systemctl enable k0sworker
systemctl start k0sworker
```

Con esto lo que se hace es habilitar el worker en el inicio incluyendo el token en el k0sworker.service para su conexión automática.

El k0sworker.service es la parte que Debian ejecutará al inicio del sistema.

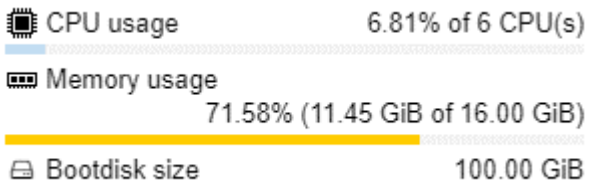
Recursos y consumo real:



2.2.5 MINIFABRIC

También ejecuta el SO (Sistema Operativo) Debian. Lo único necesario para utilizar Minifabric es Docker, y el propio binario de Minifabric, por lo que solamente corre Docker, el binario de Minifabric lo que hace es correr un contenedor Docker con el software.

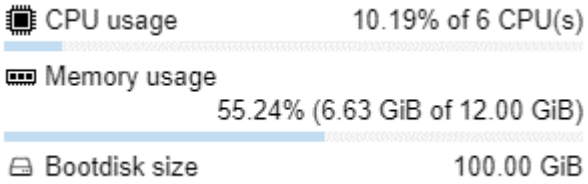
Recursos y consumo de recursos real:



2.2.6 MASTER GITLAB

Esta máquina ejecuta Debian y una instancia completa de GitLab EE.

Recursos y consumo de recursos real:

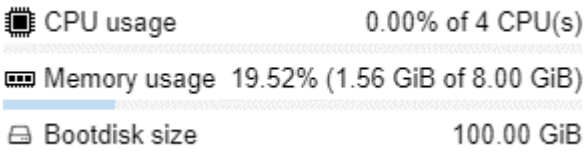


2.2.7 RUNNER GITLAB

También con Debian y ejecutando Docker junto con un worker de GitLab. El orker GitLab es el encargado de ejecutar los procesos de Ci/CD

Los workers de GitLab utilizan Docker para correr las operaciones de CI/CD

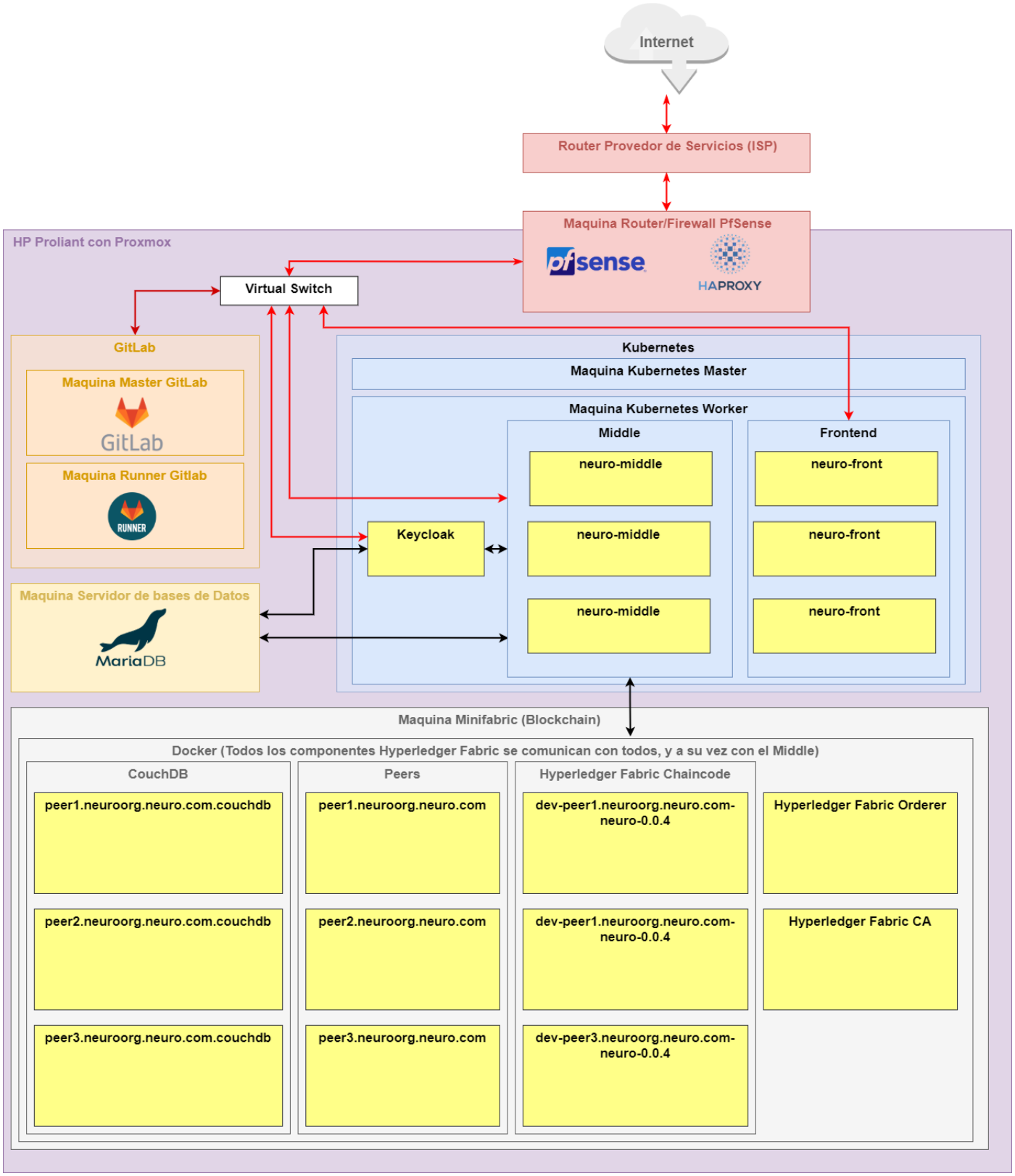
Recursos y consumo de recursos real:



El consumo de recursos aumentará cuando se ejecuten las tareas de Ci/CD, por ello ahora apenas se utilizan.

2.3 INFRAESTRUCTURA DE CONTENEDORES

En realidad, esta es la infraestructura real dado que todo se corre en contenedores, ya sea bajo Kubernetes o directamente en Docker como se hace en la blockchain.



En la imagen anterior se pueden ver coloreados en amarillo cada uno de los contenedores que son necesarios y existen en el despliegue de demostración de Neuro.

Las flechas de color rojo hacen referencia a comunicaciones de red que entran desde internet y en negro las que en ningún caso salen ni entran a o desde internet. Como se puede ver, la blockchain y la base de datos están completamente aisladas de internet, esto aporta un punto extra de seguridad.

En el caso de la blockchain, todos los contenedores se comunican entre si, y por ello se ha omitido en el esquema anterior.

2.3.1 MIDDLE, KEYCLOAK Y FRONTEND (KUBERNETES)

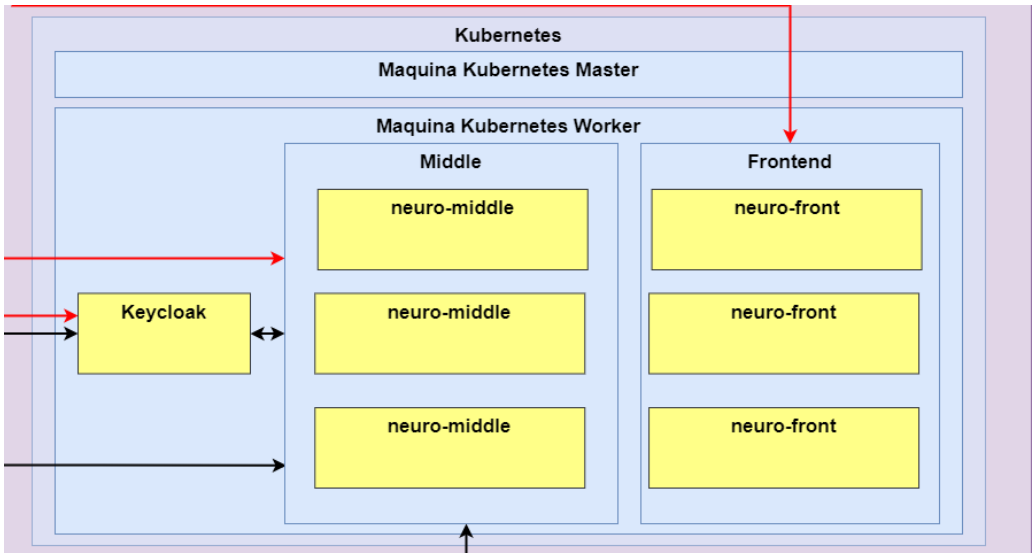
Para aportar otro punto de vista, además del esquema, a continuación se puede ver una captura con los Pods en ejecución desde Lens:

Name	Names...	Contai...	Restarts	Control...	Node	QoS	A...	Sta...
keycloak-866cbc447-n...	neuro		0	ReplicaSet	k8s-worker-1	BestEff...	29d	Running
neuro-front-6d6df8c77...	neuro		0	ReplicaSet	k8s-worker-1	Burstable	6d2...	Running
neuro-front-6d6df8c77...	neuro		0	ReplicaSet	k8s-worker-1	Burstable	6d2...	Running
neuro-front-6d6df8c77...	neuro		0	ReplicaSet	k8s-worker-1	Burstable	6d2...	Running
neuro-middle-bb5d4d9...	neuro		0	ReplicaSet	k8s-worker-1	BestEff...	6d2...	Running
neuro-middle-bb5d4d9...	neuro		0	ReplicaSet	k8s-worker-1	BestEff...	6d2...	Running
neuro-middle-bb5d4d9...	neuro		0	ReplicaSet	k8s-master-1	BestEff...	6d2...	Running

En la imagen anterior se pueden ver todos los Pods (mínima instancia de Kubernetes):

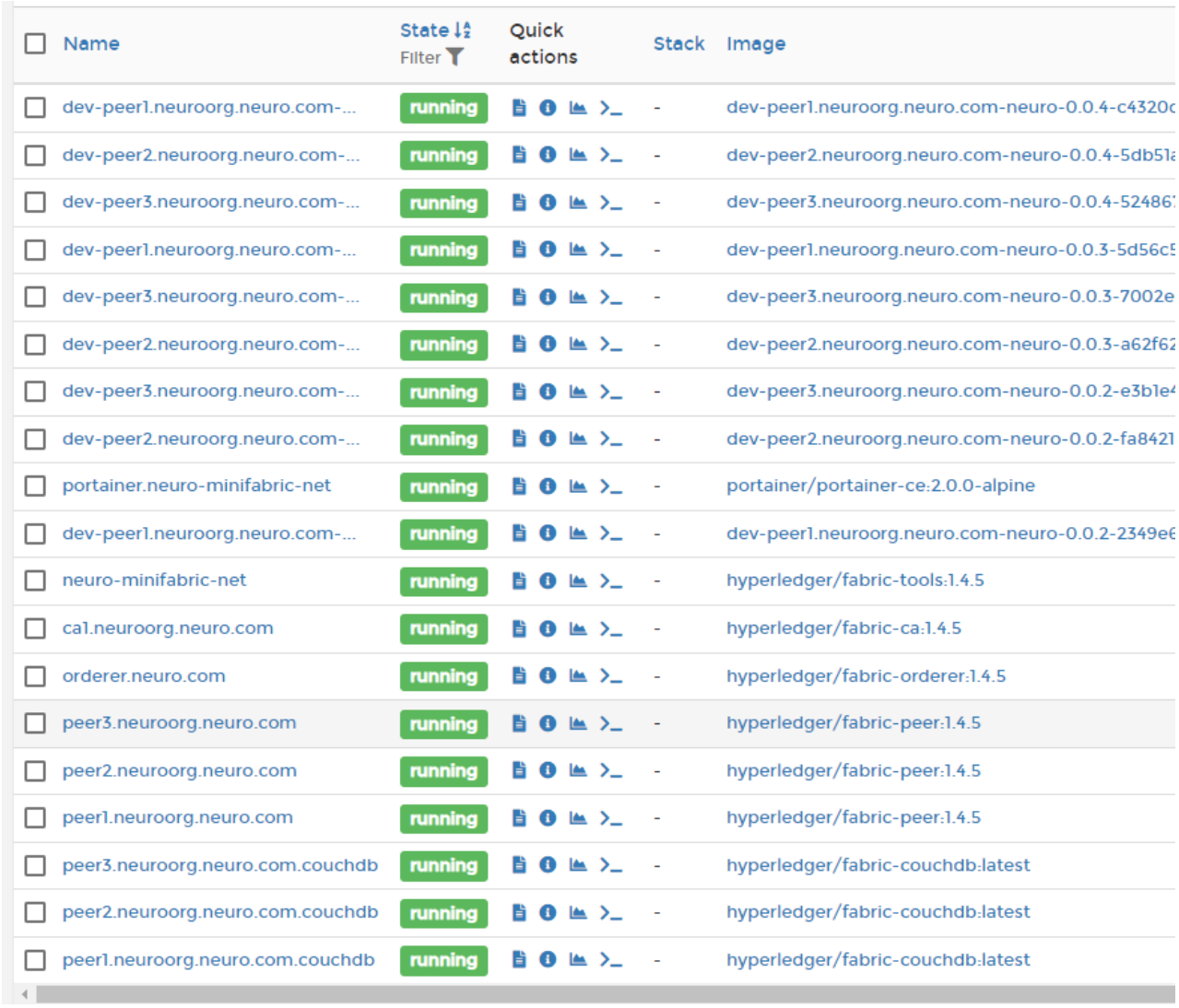
- 3 Pods Middle
- 3 Pods Frontend
- 1 Pod Keycloak

En la siguiente imagen se ha recortado la parte a la que se hace referencia del esquema:


Kubernetes
Maquina Kubernetes Master
Maquina Kubernetes Worker
Middle
Frontend
neuro-middle
neuro-front
Keycloak
neuro-middle
neuro-front
neuro-middle
neuro-front

2.3.2 BLOCKCHAIN (DOCKER PURO)

La blockchain tiene algo más de complejidad.



Name	State / Filter	Quick actions	Stack	Image
dev-peer1.neuroorg.neuro.com-...	running		-	dev-peer1.neuroorg.neuro.com-neuro-0.0.4-c4320c
dev-peer2.neuroorg.neuro.com-...	running		-	dev-peer2.neuroorg.neuro.com-neuro-0.0.4-5db51a
dev-peer3.neuroorg.neuro.com-...	running		-	dev-peer3.neuroorg.neuro.com-neuro-0.0.4-52486
dev-peer1.neuroorg.neuro.com-...	running		-	dev-peer1.neuroorg.neuro.com-neuro-0.0.3-5d56c5
dev-peer3.neuroorg.neuro.com-...	running		-	dev-peer3.neuroorg.neuro.com-neuro-0.0.3-7002e
dev-peer2.neuroorg.neuro.com-...	running		-	dev-peer2.neuroorg.neuro.com-neuro-0.0.3-a62f62
dev-peer3.neuroorg.neuro.com-...	running		-	dev-peer3.neuroorg.neuro.com-neuro-0.0.2-e3b1e4
dev-peer2.neuroorg.neuro.com-...	running		-	dev-peer2.neuroorg.neuro.com-neuro-0.0.2-fa8421
portainer.neuro-minifabric-net	running		-	portainer/portainer-ce:2.0.0-alpine
dev-peer1.neuroorg.neuro.com-...	running		-	dev-peer1.neuroorg.neuro.com-neuro-0.0.2-2349e6
neuro-minifabric-net	running		-	hyperledger/fabric-tools:1.4.5
ca1.neuroorg.neuro.com	running		-	hyperledger/fabric-ca:1.4.5
orderer.neuro.com	running		-	hyperledger/fabric-orderer:1.4.5
peer3.neuroorg.neuro.com	running		-	hyperledger/fabric-peer:1.4.5
peer2.neuroorg.neuro.com	running		-	hyperledger/fabric-peer:1.4.5
peer1.neuroorg.neuro.com	running		-	hyperledger/fabric-peer:1.4.5
peer3.neuroorg.neuro.com.couchdb	running		-	hyperledger/fabric-couchdb:latest
peer2.neuroorg.neuro.com.couchdb	running		-	hyperledger/fabric-couchdb:latest
peer1.neuroorg.neuro.com.couchdb	running		-	hyperledger/fabric-couchdb:latest

En la imagen anterior se pueden ver todos los contenedores Docker. En realidad, no todos están en uso, ya que Minifabric guarda todas las versiones del chaincode (hasta el reinicio completo de la máquina host).

También se puede ver un contenedor con nombre "portainer.neuro-minifabric-net" que hace referente a una herramienta para monitoreo.

3 ENTORNO BLOCKCHAIN IDEAL Y PRESUPUESTO

Como ya se ha ido desarrollando en todo el anexo, todos los sistemas están desplegados en Kubernetes, por lo que es bastante sencillo llevarlos a cualquier entorno de cualquier dimensión.

El problema aquí es la blockchain, dado su número de nodos y su complejidad, es difícil hacer un despliegue de grandes dimensiones y que no requiera de un equipo pendiente del funcionamiento (Esto no ocurre con Kubernetes, de lo único que hay que preocuparse es de mantener el clúster, que no es muy complejo, e incluso se resuelve fácilmente con una empresa de servicios cloud).

Para un entorno real de producción, conviene adquirir una licencia de IBM Blockchain Platform para sistemas externos a la IBM Cloud, o usar su propia Cloud.

Para entender la tarificación hay que saber que se cobra por un lado el uso del software y por otro el clúster Kubernetes u Openshift.

- **Costo mínimo de licencia:** pidiendo un presupuesto al departamento de ventas de IBM, me han hecho saber que el precio mínimo que cobran por una licencia de IBM Blockchain Platform son **25.000 dólares anuales** y permite el uso en cualquier cluster Obenshift o Kubernetes.
- **Costo mínimo SaaS (En IBM Cloud):** 0.29 por CPU asignada a la hora. Es decir, 0.29 * 24 h * 30 dias mes * 12 meses, **2505.6 dólares al año**, a lo que hay que sumar el precio del clúster Kubernetes en la IBM Cloud, **no es posible usar un clúster externo**

Las ventajas de usar la licencia es que permite usar la plataforma en cualquier clúster Kubernetes, por lo que escalarlo es más barato.

ANEXO IV - MONITOREO

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En el presente documento se describirán las diferentes herramientas usadas para monitorear toda la infraestructura.

Dado que se trata de una infraestructura compleja y con muchos módulos y nodos, es complejo monitorear utilizando los típicos ficheros de logs, es por ello que NEuro utiliza herramientas específicas de monitoreo.

2 MONITOREO GENERAL DE DISPONIBILIDAD DE SERVICIOS

Para monitorear la disponibilidad de los servicios se utiliza Statping.

2.1 STATPING

Como esto es un servicio de monitoreo enfocado al usuario final es importante que este desplegado en máquinas o servicios cloud distintos de los que se usan para el resto de los sistemas. De esta manera si todos los sistemas colapsan y dejan de estar disponibles el servicio de monitoreo seguirá disponible.

Con Statping se ha creado un panel simple en el que se puede ver la disponibilidad de los diferentes servicios relevantes para el usuario:

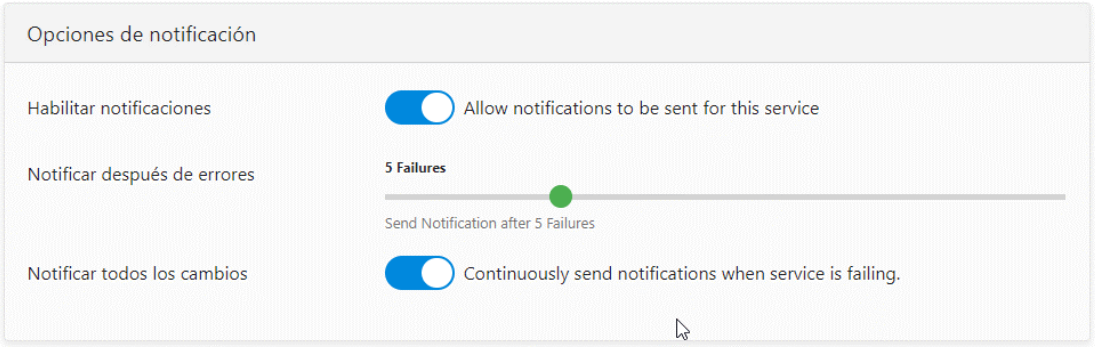
- Web
- Middle
- Servicio de identidades (Auth Server Keycloak)

En el caso de la blockchain, como es una parte transparente para el usuario no se añade en el panel de Statping.

2.1.1 NOTIFICACIONES

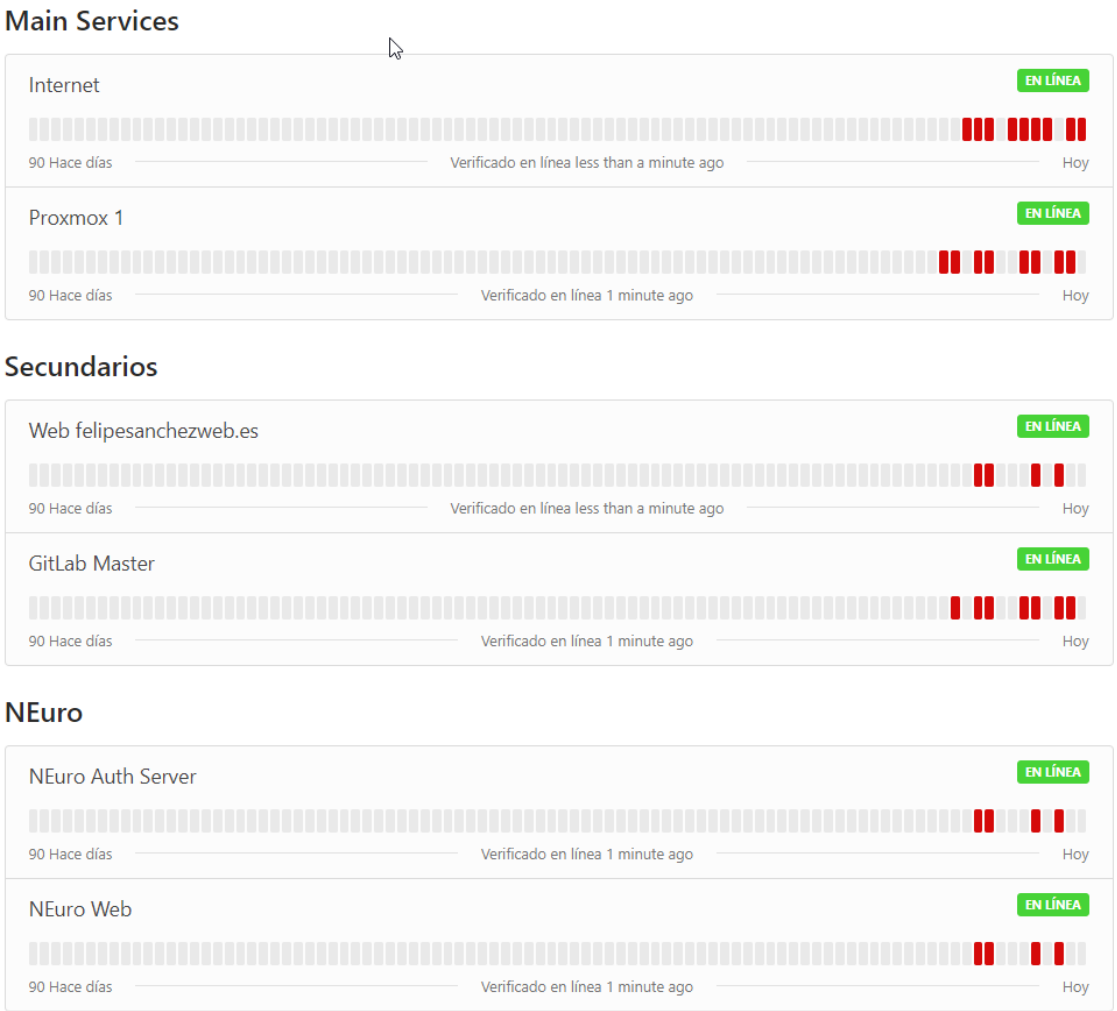
Adicionalmente, se utiliza Pushover para recibir notificaciones en caso de que alguno de los servicios deje de estar disponible. De esta manera los técnicos encargados de los servicios pueden recibir notificaciones y atenderlas lo antes posible si algo va mal.

Un pequeño ejemplo de la configuración de las alertas:



Esto especifica que cuando la comprobación falla 5 veces se enviarán las alertas.

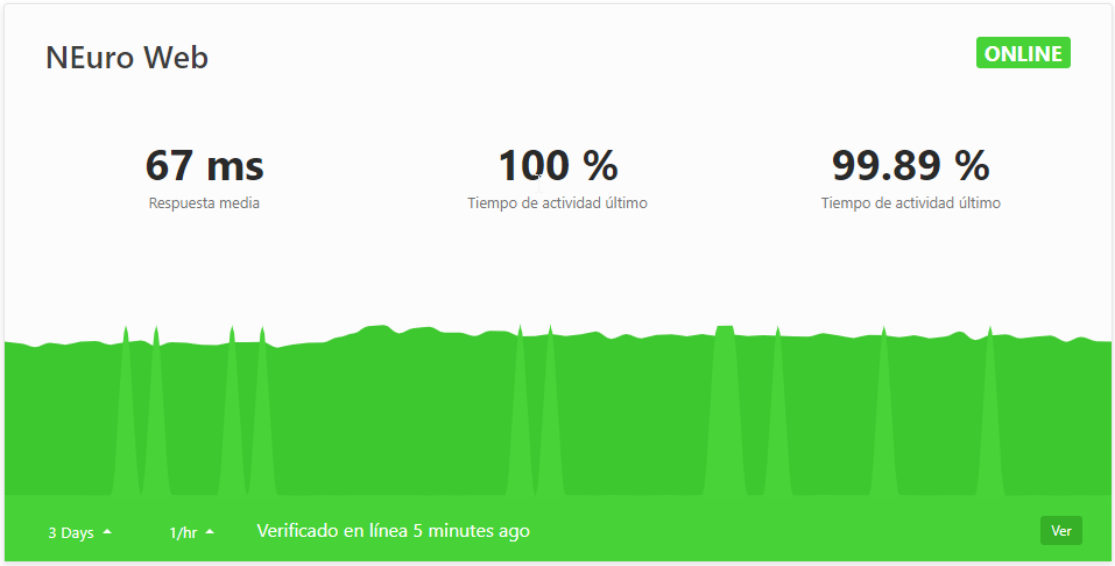
El aspecto que tiene el panel de statping es el siguiente:



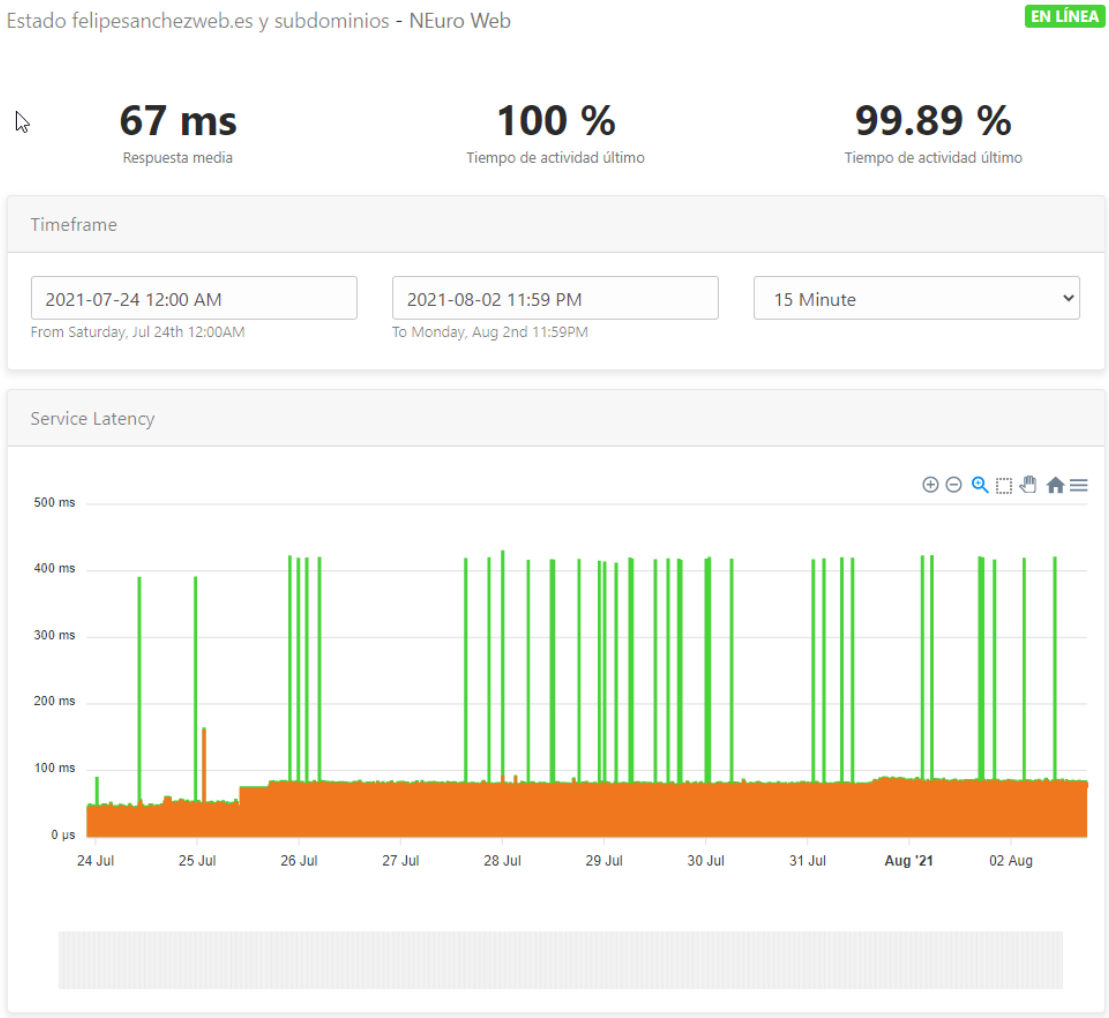
En rojo se ven los días en los que ha habido algún fallo, en este caso como el sistema está en desarrollo y desplegándose se van a ver fallos a diario, ya sea por reinicio de máquinas virtuales, del servidor completo, etc.

También se han configurado graficas de tiempo de respuesta, útiles para ver como de saturado está el sistema y los servicios relacionados.

A continuación, se puede ver una de esas gráficas:



Para terminar, también podemos ver las gráficas mucho más detalladas si hacemos click en el botón "Ver" de la gráfica general:



3 MONITOREO Y ANÁLISIS DEL CLUSTER KUBERNETES

El cluster de Kubernetes es un elemento clave en el sistema, y es por eso que se utilizan dos formas de monitoreo. Hay que destacar que también son parte del monitoreo las métricas, gestionadas por Prometheus.

Una muy buena línea de trabajo futura en cuanto a métricas podría ser presentar datos concretos de NEuro (desde el punto de vista del negocio), por ejemplo:

- Número de usuarios
- Tiempo de respuesta a las diferentes llamadas REST
- Montos transmitidos por transferencia bancaria
- Errores de los pods y de los nodos de la blockchain.
- Numero de transferencias bancarias
- Etc.

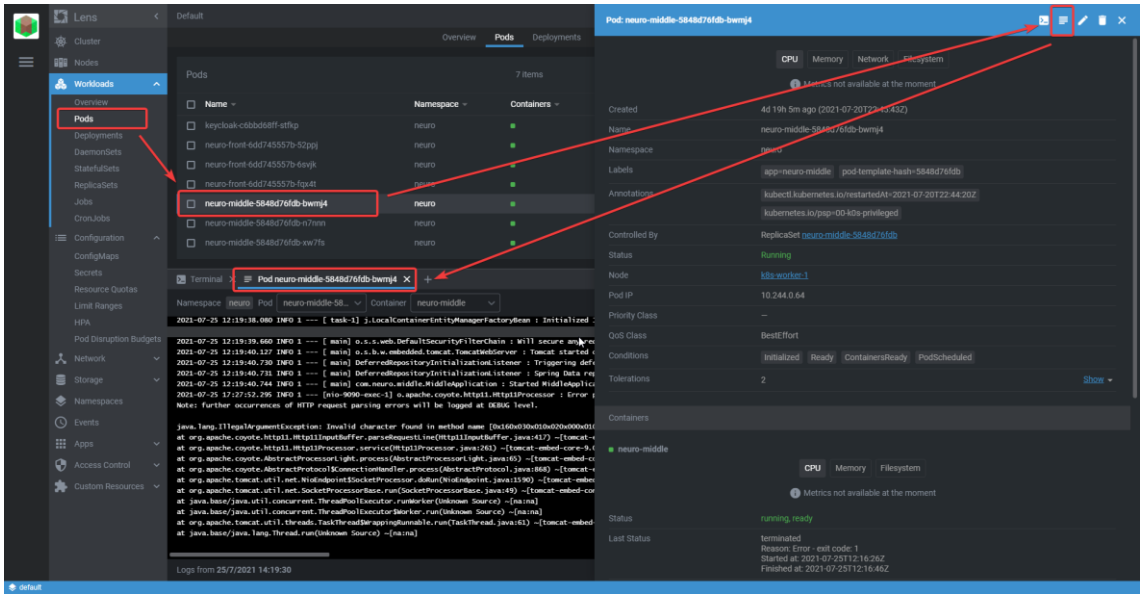
Para poder almacenar todo eso, seria ideal mandar desde el middle todas las peticiones a alguna base de datos de código de tiempo, o incluso mejor aún, a una cola (RabbitMQ por ejemplo) y desde ahí consumir los mensajes desde donde se necesiten, en este caso consumirlos para añadirlos a una base de datos como InfluxDB. Esto elimina el impacto de rendimiento que pueda tener añadir más lógica en el middle.

3.1 LENS

Con Lens se monitorea de forma manual el cluster.

Para ver los logs de cada Pod es necesario acceder a cada pod.

A continuación, se puede ver un ejemplo de como se muestran los logs del pod:



En la imagen anterior se puede ver como hay una excepción en los logs, en este caso es una excepción forzada de ejemplo.

Lens es al fin y al cabo una herramienta que hace todo lo que hace kubectl (kubectl es la herramienta usada para gestionar claustres de Kubernetes por la línea de comandos)

Esta forma puede ser útil para desarrolladores, pero para un monitoreo más complejo puede ser algo tediosa, sobre todo con más de una réplica como es el caso.

3.2 PROMETHEUS

Prometheus es una herramienta que permite obtener métricas del clúster Kubernetes.

Dispone de una pequeña interfaz gráfica web, pero es muy limitada y no se usará.

3.3 GRAFANA

Grafana ofrece un avanzado sistema para observar métricas que se obtienen desde fuentes externas de datos. Ofrece muchísimos tipos de fuentes de datos distintas: bases de datos SQL, bases de datos basadas en tiempo, etc.

Otro de los puntos fuertes es la capacidad de exportar e importar Dashboards (que son los paneles en los que se visualizan los datos). En NEuro se utilizan los paneles:

- Prometheus 2.0 Overview
- Kubernetes cluster monitoring (via Prometheus)

Es posible acceder a los paneles desde esta URL utilizando las el usuario "**viwer@viwer.viwer**" y contraseña "**+Q3zVEy6?L**" ambos sin las comillas.

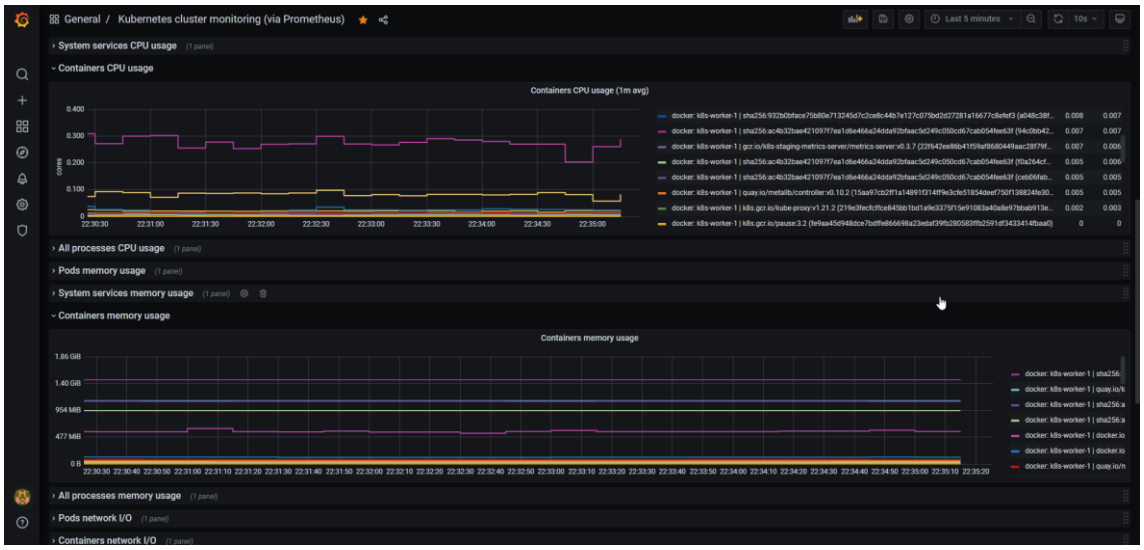
3.3.1 PROMETHEUS 2.0 OVERVIEW

Este panel proporciona métricas sobre el propio Prometheus. Uptime, errores, tiempo de consulta, etc.

3.3.2 KUBERNETES CLUSTER MONITORING (VIA PROMETHEUS)

Se trata de un panel que expone información sobre CPU, memoria, almacenamiento, red, entre otras métricas del Cluster Kubernetes Completo.

A continuación, se exponen unas imágenes de ejemplo:



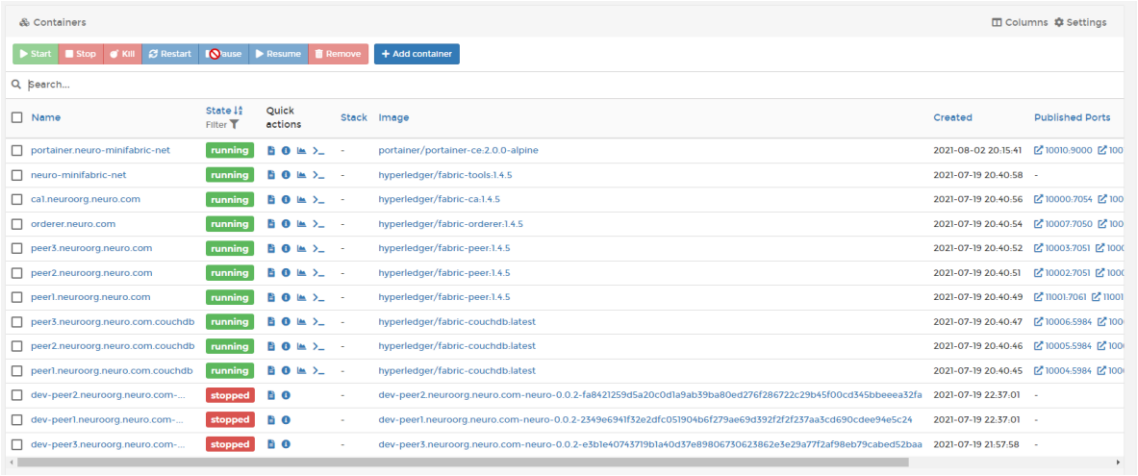
4 MONITOREO DE LA BLOCKCHAIN

Como ya viene siendo habitual, la blockchain no utiliza la misma forma de monitoreo que el resto de módulos, tiene su propia forma de monitoreo independiente.

Se usa Portainer se usa para monitorear los contenedores de la blockchain.

No se da acceso a ella dado que es peligroso manipularla, solamente para uso interno.

A modo de ejemplo se ha tomado una captura de pantalla de una situación en la que los contenedores del chaincode (código de la blockchain o Smart Contracts) se encuentran en estado "parado":

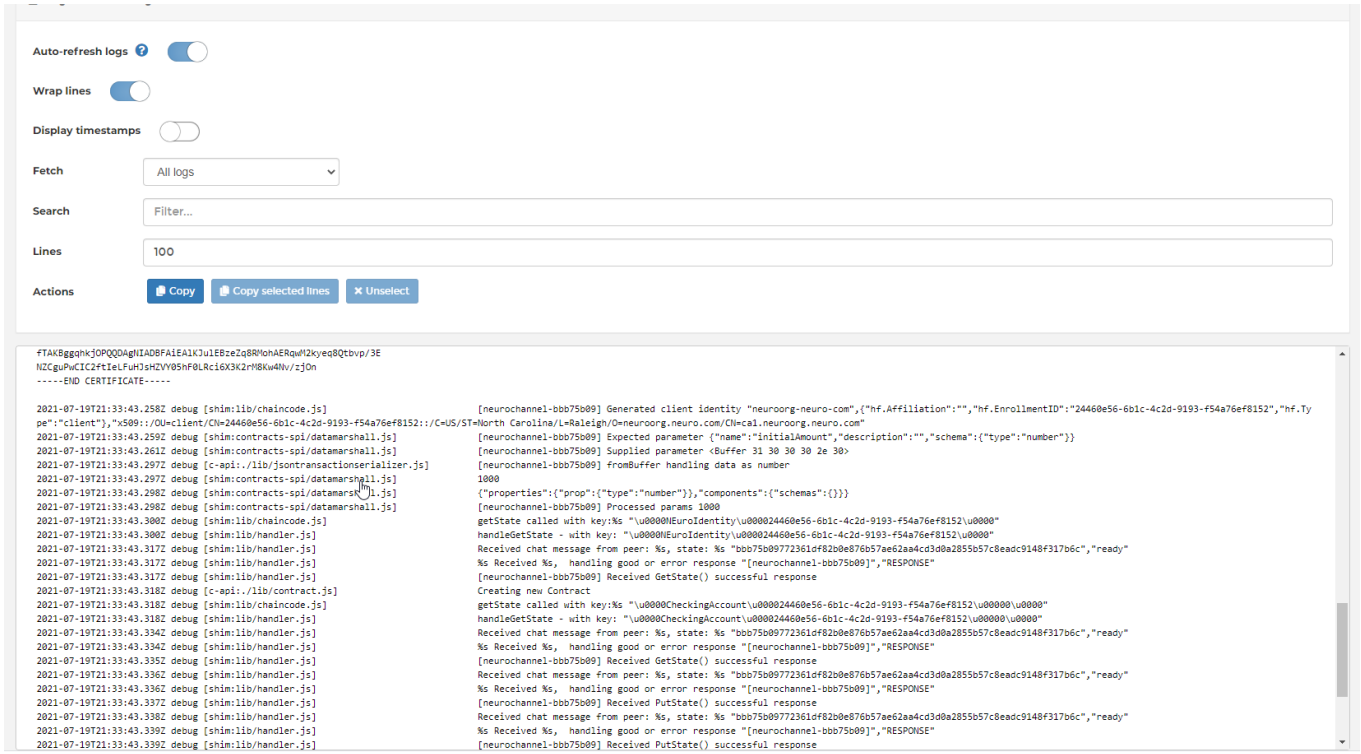


En el caso de la imagen anterior el error ha sido forzado apagando forzosamente la máquina virtual de la blockchain y no iniciando esos contenedores de nuevo.

La solución a ese error seria sencilla, volver instalar e instanciar el chaincode:

```
minifab install -n neuro -v x.x.x -l node
minifab instantiate -p ""
```

De manera similar a como se hace con Lens, con Portainer podemos acceder a los logs de cada contenedor para poder revisar los posibles fallos:



La imagen anterior muestra los logs de uno de los peers. Concretamente se puede ver una operación de creación de una nueva identidad.

De igual manera que Lens es una interfaz gráfica de kubectl CLI, Portainer es una interfaz gráfica para las operaciones que se pueden hacer con Docker CLI.

ANEXO V - CI-CD

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En este anexo se describirá detalladamente el funcionamiento de la Integración Continua y el Despliegue Continuo.

2 GITLAB

GitLab es el núcleo de toda CI/CD empleada en este proyecto.

GitLab permite montar una infraestructura que soporta flujos (Pipelines) para automatizar todo tipo de tareas, en este caso, tres tareas principales:

- Generación de imágenes Docker
- Subida de imágenes Docker a Dockerhub
- Actualización de las imágenes en los deployments de Kubernetes

3 EL CÓDIGO

En realidad todo esto no se hace desde código como tal. Se hace en su mayor parte desde descripciones en archivos YAML.

Todo esto es posible gracias a las imágenes Docker, ya que permiten encapsular toda una aplicación (Incluyendo su servidor web, etc) en un contenedor desplegable en casi cualquier parte

3.1 GENERACIÓN DE IMÁGENES DOCKER

Este es el primer paso, y se define a través de Docker files que dan lugar a imágenes docker.

Una vez comprendamos como se generan las imágenes Docker, ya podemos entrar en el funcionamiento de las etapas de los Pipelines

3.1.1 FRONTEND ANGULAR

Básicamente, en la etapa primera se compila la aplicación angular para producción, y en la segunda se mueven los artefactos generados para ser servidor por NGINX

```
# Stage 0, "build-
stage", based on Node.js, to build and compile the frontend
FROM node:14.15.4 as build-stage
WORKDIR /app
COPY ./ /app/
RUN npm install
ARG configuration=production
```

```
RUN npm run build -- --output-path=./dist/out --
configuration $configuration

# nginx state for serving content
FROM nginx:alpine
# Remove default nginx static assets
RUN rm -rf  /usr/share/nginx/html
# Copy static assets from builder stage
COPY --from=build-stage /app/dist/out/ /usr/share/nginx/html/
COPY ./nginx-custom.conf /etc/nginx/conf.d/default.conf
```

Configuración NGINX usada:

```
server {
  listen 80;
  sendfile on;
  default_type application/octet-stream;

  gzip on;
  gzip_http_version 1.1;
  gzip_disable      "MSIE [1-6]\.";
  gzip_min_length   256;
  gzip_vary         on;
  gzip_proxied      expired no-cache no-store private auth;
  gzip_types        text/plain text/css application/json application/java
script application/x-
javascript text/xml application/xml application/xml+rss text/javascript;
  gzip_comp_level   9;

  root /usr/share/nginx/html;

  location / {
    try_files $uri $uri/ /index.html =404;
  }
}
```

Aquí lo mas importante es que se redireccionan las peticiones al index.html que contiene toda la aplicación Angular. Esto se hace por que Angular tiene su propio enrutador, y las rutas URL las gestiona el internamente.

3.1.2 MIDDLE

En este caso SpringBoot ya nos facilita mucho el trabajo, ya que basta con ejecutar:

```
mvn spring-boot:build-image -Dspring-boot.build-
image.imageName=felipesanchez98/neuro-middle:latest
```

Ese comando ya genera una imagen Docker completa y lista para producción.

3.2 DESCRIPCIÓN DE LOS PIPELINES

Los pipelines en GitLab CI/CD es la manera de definir los procesos.

A continuación se procederá a explicar lo que hacen cada uno de los archivos.

Hay que tener en cuenta que solo puede existir un archivo, aunque desde ese archivo se pueden referencias otros. Esta referencia no es mas que una inclusión, es decir, es como si el archivo que se referencia se pegara en la línea del archivo padre.

En todos los pasos se puede ver:

```
    only:
        - env-develop
```

Esto hace referencia a que el pipeline se va a ejecutar únicamente con subidas en la rama env-develop

3.2.1 ARCHIVO USADO PARA EL FRONTEND ANGULAR

Este archivo consta de tres etapas:

- **Install:** instala las dependencias (npm install). Solo sirve para el paso de Tests. Requiere hacer click en un botón manualmente
- **Tests:** Ejecuta los test unitarios Jasmine. También es manual
- **Front-build-and-push-image:** es la única etapa automática y se ejecuta cuando se hace el push al repositorio git. En esta etapa es donde realmente se genera y se sube la imagen docker que anteriormente he explicado como se genera.

```
#include:
#    - template: Code-Quality.gitlab-ci.yml
#    - template: SAST.gitlab-ci.yml
#    - template: Container-Scanning.gitlab-ci.yml
#    - template: Dependency-Scanning.gitlab-ci.yml

install:
    image: node:14.15.4
    stage: install
    when: manual
    before_script:
        - cd front/Web/Angular/NEuro-PrimeNG-Freya/
    script:
```

```
        - npm install
    artifacts:
        expire_in: 1h
        paths:
            - front/Web/Angular/NEuro-PrimeNG-Freya/node_modules/
    cache:
        paths:
            - front/Web/Angular/NEuro-PrimeNG-Freya/node_modules/
    only:
        - env-develop


tests:
    image: node:14.15.4
    stage: test
    when: manual
    variables:
        CHROME_BIN: google-chrome
    dependencies:
        - install
    before_script:
        - apt-get update && apt-get install -y apt-transport-https
        - wget -q -O - https://dl-
ssl.google.com/linux/linux_signing_key.pub | apt-key add -
        - sh -
c 'echo "deb https://dl.google.com/linux/chrome/deb/ stable main" >> /etc
/apt/sources.list.d/google.list'
        - apt-get update && apt-get install -y google-chrome-stable
        - npm install -g @angular/cli
        - cd front/Web/Angular/NEuro-PrimeNG-Freya/
    script:
        - npm run test:ci
    coverage: '/Statements.*?(\d+(?:\.\d+)?)%/'
    only:
        - env-develop

front-build-and-push-image:
    stage: build-and-push-image
    image: docker:19.03.12
    services:
        - docker:19.03.12-dind
    before_script:
        - cd front/Web/Angular/NEuro-PrimeNG-Freya/
    script:
        - docker login -u $DOCKER_HUB_USERNAME -p $DOCKER_UB_PASSWORD
        - docker build -t $IMAGE_NAME .
        - docker push $IMAGE_NAME
    tags:
        - docker
```

```
    only:
        - env-develop
```

3.2.2 ARCHIVO USADO PARA EL MIDDLE

En este caso no existe ni la etapa install ni la etapa de tests unitarios.

La etapa **middle-build-and-push-image** se encarga de generar la imagen docker y subirla dockerhub

```
middle-build-and-push-image:
  stage: build-and-push-image
  image: docker:19.03.12
  services:
    - docker:19.03.12-dind
  before_script:
    - apk add --no-cache openjdk11 bash maven
    - cd "middle/API REST/SpringBoot/NEuro/"
  script:
    - docker login -u $DOCKER_HUB_USERNAME -p $DOCKER_UB_PASSWORD
    - mvn spring-boot:build-image -Dspring-boot.build-
image.imageName=felipesanchez98/neuro-middle:latest
    - docker push felipesanchez98/neuro-middle:latest
  tags:
    - docker
  only:
    - env-develop
```

3.2.3 ARCHIVO PADRE GENERAL

Este archivo se encarga de:

- Incluir los dos archivos anteriores
- Hacer una instalación temporal de Kubectl
- Aplicar los archivos YAML que describen los deployments de Kubernetes
- Reiniciar los Pods de Kubernetes para que actualicen sus imágenes docker (Y con ellas el nuevo codigo)

```
image: docker:19.03.12
services:
```

```
  - docker:19.03.12-dind

stages:
    - install
    - test
    - build-and-push-image
    - deploy-in-k8s

variables:
    DOCKER_TLS_CERTDIR: "/certs"
    IMAGE_NAME: felipesanchez98/neuro-front:latest


include:
  - local: '/middle/API REST/SpringBoot/NEuro/.gitlab-ci.yml'
  - local: '/front/Web/Angular/NEuro-PrimeNG-Freya/.gitlab-ci.yml'


deploy-in-k8s:
  image: bash:latest
  stage: deploy-in-k8s
  before_script:
    - apk --no-cache add dumb-init gettext ca-certificates openssl
    - wget https://storage.googleapis.com/kubernetes-
release/release/$(wget https://storage.googleapis.com/kubernetes-
release/release/stable.txt -q -O -)/bin/linux/amd64/kubectl -q -
O /usr/local/bin/kubectl
    - chmod a+x /usr/local/bin/kubectl
    - apk --no-cache del ca-certificates openssl
  script:
    - echo $KUBECTL_CONFIG_FILE_CONTENT
    - export KUBECONFIG=$KUBECTL_CONFIG_FILE_CONTENT
    - kubectl apply --validate -f "front/Web/Angular/NEuro-PrimeNG-
Freya/k8s/neuro-front.yml"
    - kubectl apply --validate -
f "middle/API REST/SpringBoot/NEuro/k8s/neuro-middle.yml"
    - kubectl -n neuro rollout restart deployment.apps/neuro-front
    - kubectl -n neuro rollout restart deployment.apps/neuro-middle
  only:
    - env-develop
```

ANEXO VI – REQUISITOS Y PLANIFICACIÓN

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En este anexo se recoge todo lo relacionado a la gestión del proyecto, metodologías usadas y por qué se han usado.

2 METODOLOGÍA

La metodología seguida para hacer la gestión del proyecto ha sido en su totalidad SCRUM. Mas específicamente el framework SCRUM que ofrece la gente de Atlasian con Jira.

Jira es un software que permite la gestión integra de proyectos de todo tipo mediante metodologías Agiles.

2.1 METODOLOGÍAS AGILES

Las metodologías agiles (Agile) tienen como objetivo adaptar la forma de trabajo en el proyecto al propio proyecto. Con ello lo que se pretende conseguir es una rápida adaptación al cambio

2.2 SCRUM

SCRUM es un framework (marco de trabajo) que sigue las metodologías Agiles y da ciertas reglas que hacen mas sencillo trabajar colaborativamente en equipo.

2.2.1 ASPECTOS IMPORTANTES DE SCRUM

En esta sección se van a definir los aspectos mas importantes que hay que tener en cuenta para entender como se ha gestionado NEuro.

Backlog: Se trata de una lista de incidencias, en orden de prioridad. Mas arriba significa mas prioritaria.

Epic (Epica): Se trata de un conjunto de incidencias relacionadas. Representa una carga de trabajo relativamente grande.

Incidencia (aplicado a Jira): Es la unidad mínima de trabajo y es la pieza fundamental del trabajo. Representan Historias de Usuario, Errores, Tareas, etc. Explicada con más detalle posteriormente.

Sprint: Se trata de un intervalo de tiempo definido en el que ha de poder resolverse cualquier incidencia. Si una incidencia ocupa varios Sprints significa que tenemos que dividirla en tos mas pequeñas. El tiempo suele ser de unas pocas semanas, en NEuro cuatro semanas.

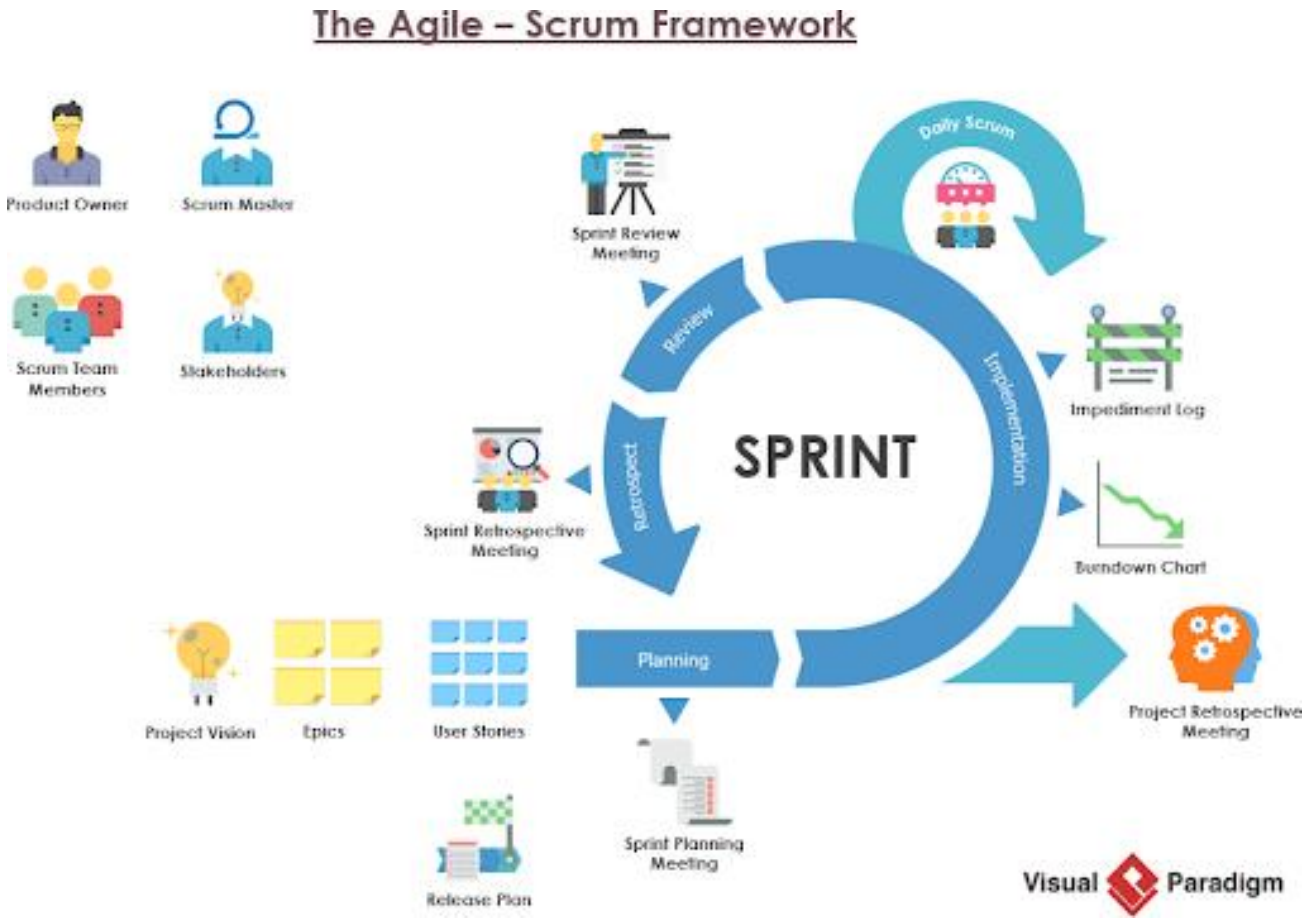
Tablero del Sprint: Se trata de una representación visual en forma de tablero Kanban del estado de las incidencias del Sprint Actual.

Roles:

- Product Owner o PO: como su nombre indica es el dueño del producto, y es el encargado de priorizar los trabajos y tareas, y de definirlos.
- Scrum Master: suele considerarse el líder del SCRUM. Genera las tareas del backlog y las prioriza con las directrices del PO. Se puede considerar la unión entre el PO y el equipo de desarrollo. Ayuda al PO con la definición correcta de los requisitos y al equipo de desarrollo con la entrega del producto.
- Equipo de desarrollo o de trabajo (Scrum Team): se trata del equipo encargado de ir resolviendo todas las tareas. Idealmente cada persona debe ser capaz de realizar el desarrollo de inicio a fin, aunque hay expertos que prefieren separar en pequeñas Squads, por ejemplo, de Desarrolladores, Ingenieros QA, diseñadores, etc.

Uno de los puntos clave de SCRUM es la transparencia del proceso, y para ello se definen una serie de **reuniones:**

- **Daily:** Reunión diaria de unos minutos en la que cada miembro mencionará sus avances, bloqueos, y la dirección que está tomando
- **Review:** al final de cada sprint. Se muestran los avances realizados en el sprint y, en algunos casos, se hace una pequeña demo del estado del producto.
- **Retrospective:** realizada al final de cada, después de la Review. En ella se analiza como



2.3 SCRUM APLICADO A JIRA

Jira se basa en SCRUM, aunque no por ello lo sigue al pie de la letra. Hay ciertas particularidades que con Jira que hay que tener en cuenta.

2.3.1 FLUJO DE LAS INCIDENCIAS

EL flujo de las incidencias es tremendamente importante dado que determina cuando se va a abordar la misma. Consta de varias etapas:

1. Adición de la incidencia en el backlog
2. Completado de la incidencia.
   2.1. Adición de comentarios
   2.2. Adición de la Épica correspondiente si procede
3. Asignación de la incidencia al responsable de su resolución
4. Asignar la incidencia a cualquier Sprint, incluido el actual
5. Monitorear como el desarrollador va pasando la incidencia de los estados To Do, In Progress, Test Ready
6. Cuando la incidencia se sitúa en Test Ready, alguien del equipo de QA ha de testear si es verdad que ha sido resuelta y en caso afirmativo pasarla a Done, en caso contrario, pasarla a In Progress.

3 LA METODOLOGÍA APLICADA EN NEURO

En NEuro he decidido hacer un pequeño recopilatorio de las mejores ideas que he visto a la hora de gestionar proyectos por las empresas que he pasado.

Para gestionar NEuro, existen tres tipos de incidencias:

- **Historia:** llamamos historia a cualquier nuevo desarrollo que implique software. También pueden llamarse HU (Historia de Usuario) o UH (User History)
- **Tarea:** las tareas son cualquier cosa que a priori no implica software y podemos resolver sin generar una nueva versión del producto. Por ejemplo, cambiar la configuración de caducidad de tokens de Keycloak.
- **Error:** estas son incidencias que han de ser resueltas cuanto antes. Se trata de errores, bugs, o simplemente algo que no funciona como se espera. Por ejemplo un usuario ha detectado que con una determinada combinación de acciones en la web dan como resultado que deje de funcionar. Puede implicar código o no.

Dado que NEuro solo tiene una persona implicada en el desarrollo por el momento, que soy yo mismo, todas las incidencias son historias de usuario que he ido resolviendo.

La principal ventaja de esta forma de gestionar el proyecto es que no es necesario que desde un principio tengamos definidas todas y cada una de las funcionalidades. Esto va a obligar a los desarrolladores a elaborar código mucho mas modular y adaptable a cambios.

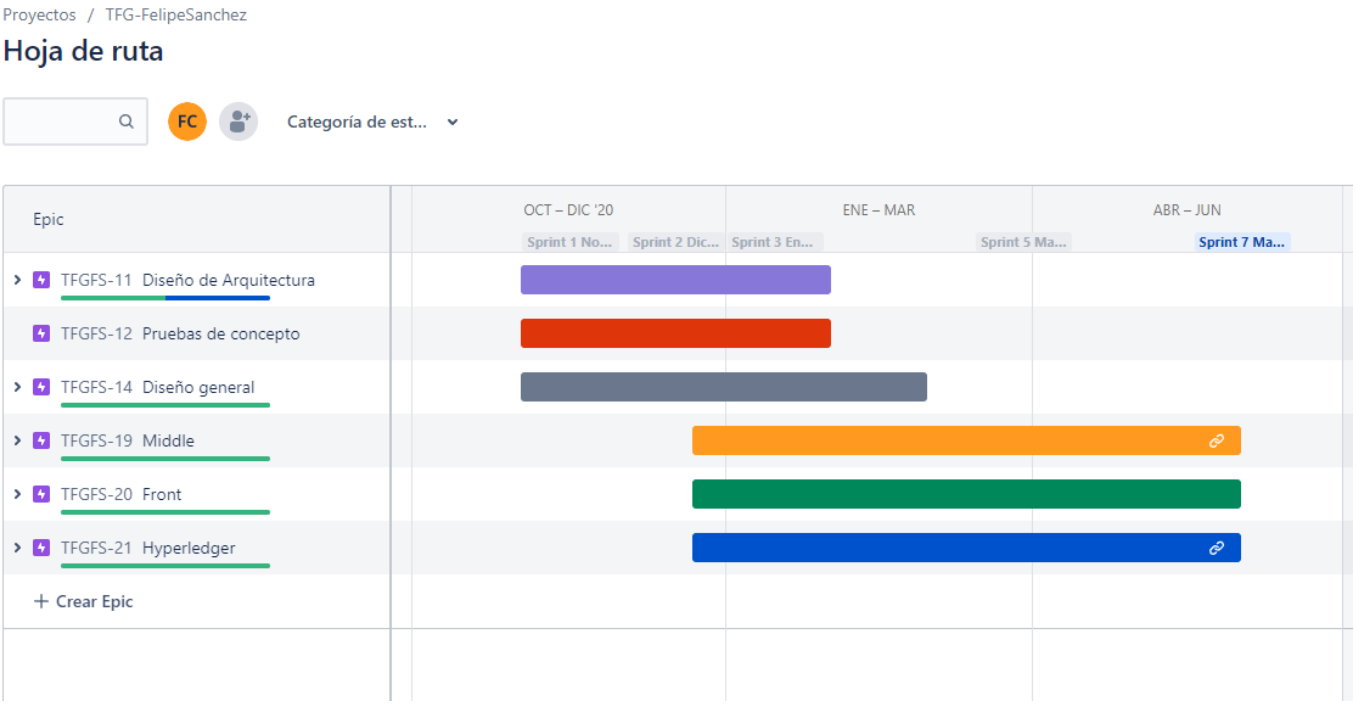
Los cambios, al igual que en SCRUM, son algo con lo que se convive, en otras metodologías los cambios implican rehacer parte del sistema o incluso el rechazo de los mismos cambios, en SCRUM y derivados, idealmente no es así.

4 REQUISITOS Y PLANIFICACIÓN

4.1 ÉPICAS

Se han creado seis épicas para todo el proyecto, y se estableció su fecha de inicio y una aproximación de la fecha final.

En la siguiente imagen se puede ver la estimación temporal:



La vista esta configurada para que se muestren tres meses por división. Lo importa aquí es ver las épicas que empiezan antes que otras.

4.1.1 DISEÑO DE ARQUITECTURA

Esta épica tenia una estimación de unos tres meses.

En ella se esperaba definir la arquitectura a alto nivel, que acabó siendo la división de Frontend, Middle y Blockchain.

Para ello se elaboraron pequeñísimas pruebas de concepto iniciales, también a alto nivel y sin detallar funcionamientos.

A continuación, se puede ver un fragmento de una de las Historias de Usuario y su documentación. Concretamente se trata de la HU "Diseño de Arquitectura de autenticación". Esta HU contiene intrínsecas varias pruebas de concepto que pertenecen a la épica de pruebas de concepto.

4.1.1.1 HU TFGFS-6

Se pedía en la HU: Diseño de Arquitectura de autenticación

El desarrollo de la HU (En este caso no implica Código, es solo un analisis):

Primera idea posiblemente no viable)

Diseño de la arquitectura para el front, hay que tener en cuenta que es un requisito la autenticación con certificados digitales (mTLS, Mutual TLS). Este requisito complica el uso de Angular, hace imprescindible usar un servidor que soporte mTLS.

A primera vista se usará NodeJS + Express Hay que hacer pruebas de concepto para ver si es viable.

Es posible que se elabore una API REST para parsear y estandarizar las interacciones con la Blockchain. Siempre teniendo en cuenta que las replicas este sistema podrá estar replicado en infinitas maquinas sin necesidad de comunicación entre ellas (Mas allá de la Blockchain).

Después de las pruebas de concepto y analisis:

La solución de usar mTLS no es posible, la api de hyperledger necesita acceso a la clave privada, dicha clave no es accesible desde el servidor.

Solución teórica 1:

Esta solución teórica es usar una InMemoryWallet y almacenar las wallet en archivos encriptados (con AES) en local.

El único momento en el que la contraseña y el material criptográfico pasa por internet seria al descargar el archivo de la wallet, esto podría incluso evitarse haciendo que el receptor de la identidad digital obtenga el archivo físicamente en un medio de almacenamiento como un USB que se otorga al acreditar su identidad en la oficina. Siempre el archivo de wallet irá completamente cifrado.

En caso de que el receptor olvide su contraseña habrá que elaborar un plan de recuperación seguro, no es tan sencillo como darle al botón "he olvidado mi contraseña" ya que la verificación de la identidad se hace presencialmente.

Solución descartada por requerir un modelo de autenticación poco común y demasiado dependiente de la administración que los usuarios hacen con las credenciales.

Solución teórica 2:

Esta solución seria usar un HSM (Modulo de hardware de seguridad). Esta solución depende de un tercero que hace el modulo de seguridad, posiblemente sea cara pero es muy segura, ofrecen la posibilidad de almacenar las claves privadas cifrando y descifrando internamente las claves privadas nunca saldrían)

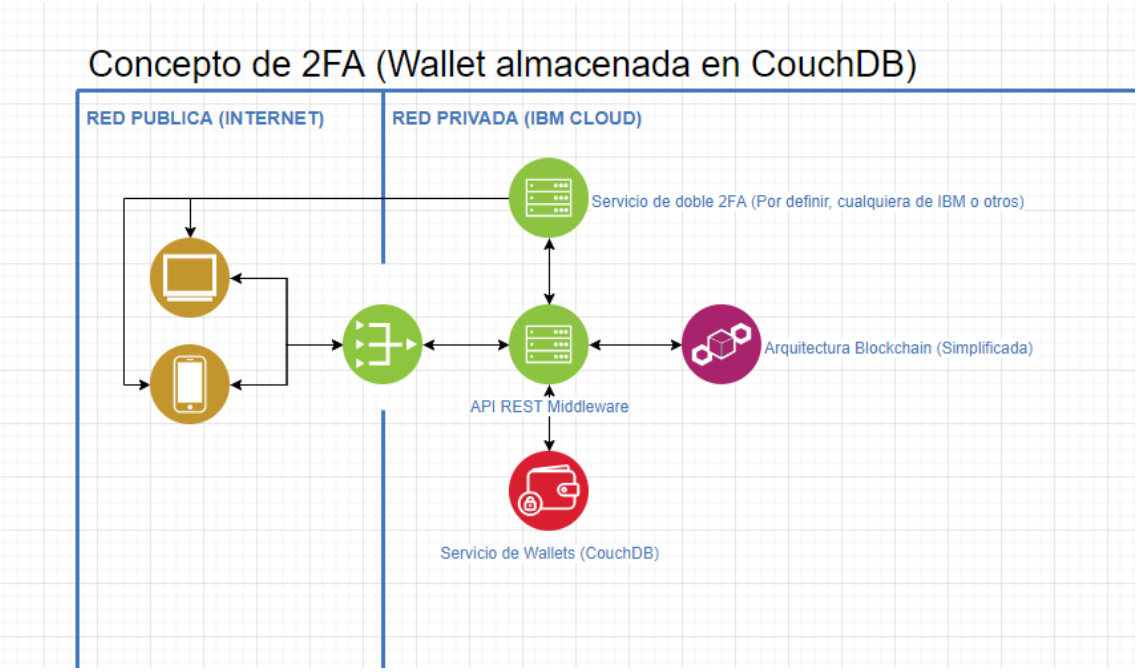
Descartada por no disponer de un HSM y por las posibles complicaciones para los usuarios

Solución teórica 3:

Uso de 2FA y un Middleware que usa una CouchDBWallet (o similar).

Después de varios días pensando posibles soluciones se plantea la ultima (?) solución y la que posiblemente sea la definitiva por su relación seguridad-sencillez para los usuarios finales.

A continuación, se adjunta la imagen de la arquitectura simplificada:



4.1.2 PRUEBAS DE CONCEPTO

En esta épica se esperaba tener un conjunto de pruebas de concepto reales con las diferentes tecnologías.

El mayor número de pruebas de concepto en realidad han sido para la elaboración del middle.

Todas estas pruebas de concepto dan lugar a la HU TFGFS-6 descrita anteriormente como ejemplo.

4.1.3 DISEÑO GENERAL

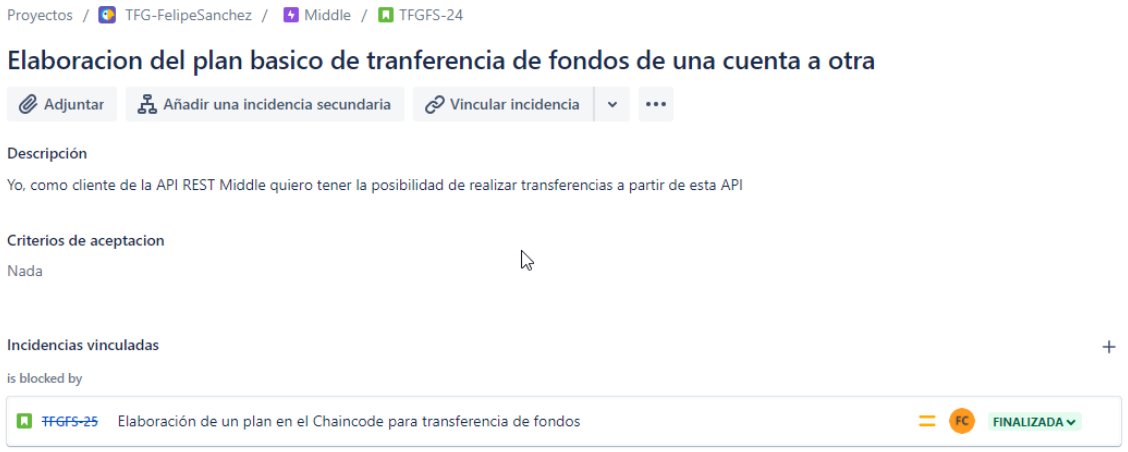
Aquí se han colocado muy pocas Historias de Usuario (HUs). Solamente la TFGFS-13 "Elaboracion de un plan para recuperar una contraseña de wallet o archivo de wallet perdidos" que finalmente se desestimó y no aplicó.

En realidad, esta épica era una especie de cajón de sastre, donde se irían metiendo HUs que no encajaran en otras por su naturaleza. Finalmente no fue necesaria.

4.1.4 MIDDLE

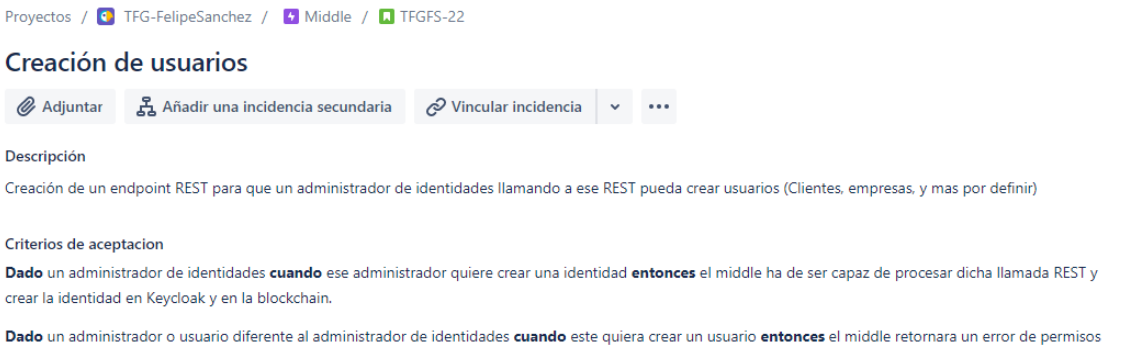
En esta Epica se han ido añadiendo las HUs correspondientes al Middle.

Dos ejemplos son la TFGFS-22 y TFGFS-24. Sus descripciones son las siguientes:



Se puede ver una dependencia en la imagen anterior. Así es como se ven las dependencias en Jira.

Lo que el desarrollador ha de hacer es revisar si las incidencias anteriores ya han sido resueltas antes de empezar el desarrollo.



Al igual que las épicas Front y Hyperledger, tiene una duración de 5 meses y medio, y su inicio se estableció un mes y medio antes de que las épicas Diseño de Arquitectura y Pruebas de concepto terminasen.

4.1.5 FRONT

De forma similar al Middle, en esta épica se han definido las HUs correspondientes al frontend Angular.

4.1.6 HYPERLEDGER

4.2 REQUISITOS POR ÉPICAS

4.2.1 DISEÑO DE ARQUITECTURA

4.2.1.1 TFGFS-6 DISEÑO DE ARQUITECTURA DE AUTENTICACIÓN

En este HU se trata de añadir en la descripción de la misma varios diseños y su posible viabilidad. Simplemente documental

4.2.1.2 TFGFS-9 ELABORACION ESQUEMAS DOCUMENTALES DISEÑO DE ARQUITECTURA FRONT (ANGULAR)

Es necesario que el arquitecto de software Angular elabore una arquitectura capaz de soportar una biblioteca amplia con componentes de diseño bonitos.

Como sugerencia la biblioteca puede ser PrimeNG junto con alguna de suis plantillas de pago.

4.2.1.3 TFGFS-10 ELABORACIÓN ESQUEMAS DOCUMENTALES DISEÑO DE ARQUITECTURA HYPERLEDGER FABRIC

Hacer un mapa conceptual o esquema visual bonito de cómo funciona esta parte del sistema y su comunicación con otras. Hacer tres:

1. Esquema conceptual de alto nivel
2. Esquema a bajo nivel. Ojo aquí, es importante dejar claro todo lo que tiene que ver a bajo nivel con HL Fabric, CouchDB,s CAs,, peers, etc.
3. Esquema de comunicaciones de este modulo con el resto. Es posible que solo sea con NodeJS + Express.

4.2.1.4 TFGFS-7 ELABORACIÓN ESQUEMAS DOCUMENTALES DISEÑO DE ARQUITECTURA HYPERLEDGER FABRIC

Definir como será la arquitectura del Chaincode y de sus Smart Contracts. Desarrollo en Typescript (SDK Node)

4.2.2 PRUEBAS DE CONCEPTO

4.2.2.1 TFGFS-40 POC KEYCLOAK CON SPRINGBOOT

Se pide hacer una prueba de concepto con código real para ver si es viable y usable en producción.

4.2.2.2 TFGFS-41 POC SPRINGBOOT CON HL FABRIC SDK Y WALLET CUSTOMIZADA (SQL)

Se pide hacer una prueba de concepto con código real para ver si es viable y usable en producción.

4.2.2.3 TFGFS-42 POC SPRINGBOOT Y ANGULAR OIDC

Se pide hacer una prueba de concepto con código real para ver si es viable y usable en producción.

4.2.3 DISEÑO GENERAL

4.2.3.1 TFGFS-13 ELABORACION DE UN PLAN PARA RECUPERAR UNA CONTRASEÑA DE WALLET O ARCHIVO DE WALLET PERDIDOS

No aplica, desestimada. Keycloak ya se encarga de eso.

4.2.4 HYPERLEDGER

4.2.4.1 TFGFS-23

Se estudiará la mejor forma de registrar usuarios en la blockchain, y mas importante, inicializar sus datos, monto de NEuros disponibles, etc.

A estas alturas aun es posible que HL Fabric 2 evolucione suficiente como para ser viable, aun así no se deberá tener en cuenta las nuevas caracteristicas de la version 2 y se hará compatible con 1.4

4.2.4.2 TFGFS-25

Se pide desarrollar el chaincode necesario para que las identidades puedan transferir fondos entre las cuentas (Checking Accounts)

4.2.4.3 FTGFS-43

Se pide crear los modelos y la lógica necesaria para que el chaincode soporte lo siguiente:

- Crear identidades de ciudadano con un monto de dinero inicial en una cuenta inicial
- Crear identidades de empresa las cuales tienen asociada una cuenta bancaria de una hacienda en la que se pagarán los impuestos.
- Crear identidades de haciendas publicas (Que van a recibir los pagos de impuestos)

4.2.4.4 TFGFS-44

Se pide diseñar el modelo de datos y la lógica para soportar desde la blockchain lo siguiente:

- Creación de cuentas bancarias
- Posibilidad de transferir fondos de una cuenta a otra
- Posibilidad de ver el historial de fondos transferidos

4.2.4.5 TFGFS-45

Se pide que se desarrolle un conjunto de modelos y de lógica de negocio en el chaincode que permita:

- Generar facturas por empresas
- Pagar facturas por empresas y por particulares (ciudadanos)
- Pago automático de impuestos al pagar la factura

4.2.5 MIDDLE

4.2.5.1 TFGFS-46

Una vez realizados los servicios de la blockchain es necesario elaborar los wrappers del middle para hacer las llamadas REST correspondientes.

4.2.6 FRONT

4.2.6.1 TFGFS-47

Dado un usuario con rol “Administrador de identidades“ ha de poder crear una nueva identidad con rol “Usuario“.

La idea es que un funcionario genere "Usuarios", siempre teniéndolos delante para identificarlos.

Campos del formulario:

- Identificador: Algo como un DNI, pasaporte, etc.
- e-Mail
- Nombre
- Apellidos
- Monto inicial: Será el dinero que recibe el funcionario al crear la identidad

Cuando se crea el usuario ha de retornarse un PDF imprimible, que el funcionario (Administrador de identidades) proporcionará al Usuario para su primer inicio de sesion.

4.2.6.2 TFGFS-48

Dado un usuario con rol "Administrador de identidades" ha de poder crear una nueva identidad con rol "Empresa".

La idea es que un funcionario genere Empresas, siempre teniendo delante al gerente de la misma.

Campos del formulario:

- Identificador: Algo como un CIF, Numero de IVA intracomunitario, etc.
- e-Mail
- Nombre: Nombre de la empresa
- Descripción corta: pequeña descripcion a modo de "apellido" de la empresa.
- Selector de la hacienda asociada
- Selector de la cuenta bancaria de esa hacienda asociada.

Cuando se crea el usuario ha de retornarse un PDF imprimible, que el funcionario (Administrador de identidades) proporcionará al gerente de la empresa para su primer inicio de sesión.

4.2.6.3 TFGFS-49

Los datos visibles en el listado de cuentas bancarias serán:

- Nombre: Que incluirá el nombre y el identificador de la cuenta, ademas de una forma de copiar el id de la cuenta bancaria
- Monto de la cuenta

- Peso: es el porcentaje de dinero que representa la cuenta, dentro del total de todas las cuentas
- Botonera de acciones:
    - Botón para ver historial
    - Botón para hacer transferencia con la cuenta de la fila

Para realizar una transferencia será necesario indicar:

- Cuenta receptora
- Monto
- Descripción

En el historial de transferencias se podrán ver los siguientes datos:

- Balance de la cuenta después de la transacción
- Monto: Si es dinero entrante en verde, si es saliente en rojo
- Descripción
- Cuenta bancaria involucrada en la transferencia

4.2.6.4 TFGFS-50

Dado un usuario de tipo empresa se pide que sea posible generar facturas.

Las facturas han de estar estructuradas y ha de poderse especificar los impuestos de cada producto individualmente.

La factura ha de contener los siguientes datos:

- Cuenta en la que se hará el ingreso (Selector de las cuentas bancariasde la empresa)
- ID de la identidad del pagador (Formato UUID)
- Descripción (Multilinea)
- Tabla de conceptos. Ha de tener la siguyientes columnas
    - Unidades
    - Condepto (Texto de una linea)
    - Precio unitario sin impuestos
    - Porcentaje de impuestos
- Precio total del concepto (unidades * precio unitario con impuestos)

En el pie de la tabla ha de venir el total de impuestos y el precio final con impuestos

4.2.6.5 TFGFS-51

Los usuarios de tipio: Usuario, Empresa y Hacienda hande poder ver las facturas que estan relacionadas con ellos.

Ha de ser posible filtrar la tabla por:

- Pendientes
- Pagadas
- Todas

Se debe poder ver el detalle completo de la factura, además del resumen que aparecerá en la propia tabla.

Desde la tabla ha de ser posible pagar una factura si no esta pagada.

5 REFERENCIAS

https://www.atlassian.com/es/agile/project-management/epics-stories-themes

https://programacionymas.com/blog/scrum-product-backlog#:~:text=El%20product%20backlog%20en%20Scrum,al%20inicio%20de%20un%20proyecto.

https://www.atlassian.com/es/software/jira/guides/getting-started/basics#step-4-create-an-issue

https://www.atlassian.com/es/agile/tutorials/sprints

ANEXO VII - MANUALES

SISTEMA MONETARIO Y ADMINISTRACIÓN PÚBLICA BASADO EN BLOCKCHAIN

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

AUTOR

Felipe Sánchez Calzada

TUTORES

Gabriel Villarrubia González

Rodrigo Santamaría Vicente

CONTENIDO

1 INTRODUCCIÓN

En este anexo están disponibles los manuales de todas las funcionalidades de la plataforma web.

Se irán describiendo en orden de roles y en cada funcionalidad se especificarán los roles que tienen acceso a ella.

2 CONCEPTOS GENERALES

Identidad: llamaremos identidad a cualquier usuario con cualquier rol.

Cuenta bancaria: son cuentas bancarias al uso, como las que puedes tener en cualquier banco, pero recordemos que las de NEuro no pertenecen a ningún banco.

Roles (De mayor a menor permiso):

- Super Usuario: Solamente disponible via servicios rest. No expone nada desde la web.
- Administrador de identidades
- Hacienda publica
- Empresa
- Usuario/Ciudadano

Aplicación TOTP: aplicación (Normalmente de móvil) que permite generar códigos de un solo uso para acceder a NEuro.

3 FUNCIONALIDAD

3.1 INICIO DE SESIÓN

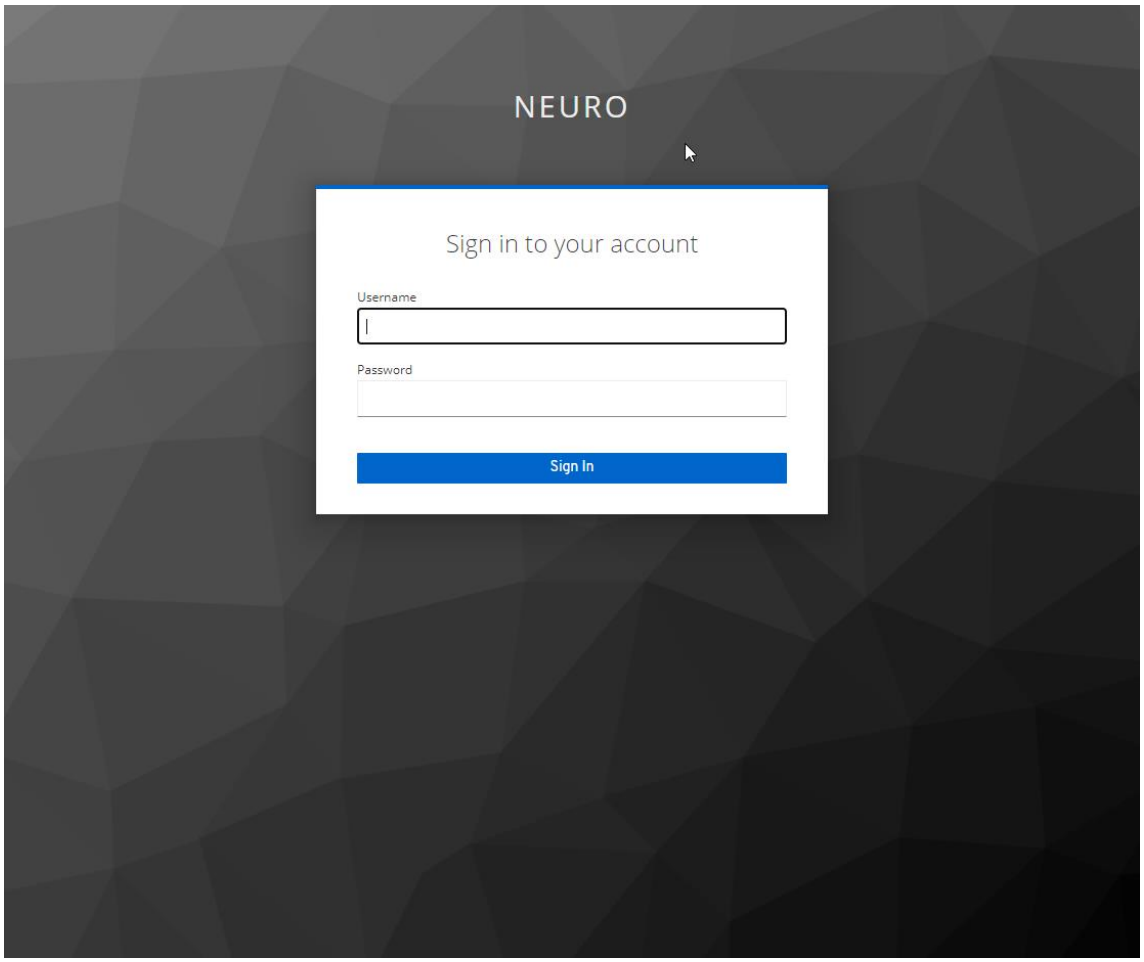
3.1.1 PRIMER INICIO DE SESIÓN

ROLES: Cualquiera

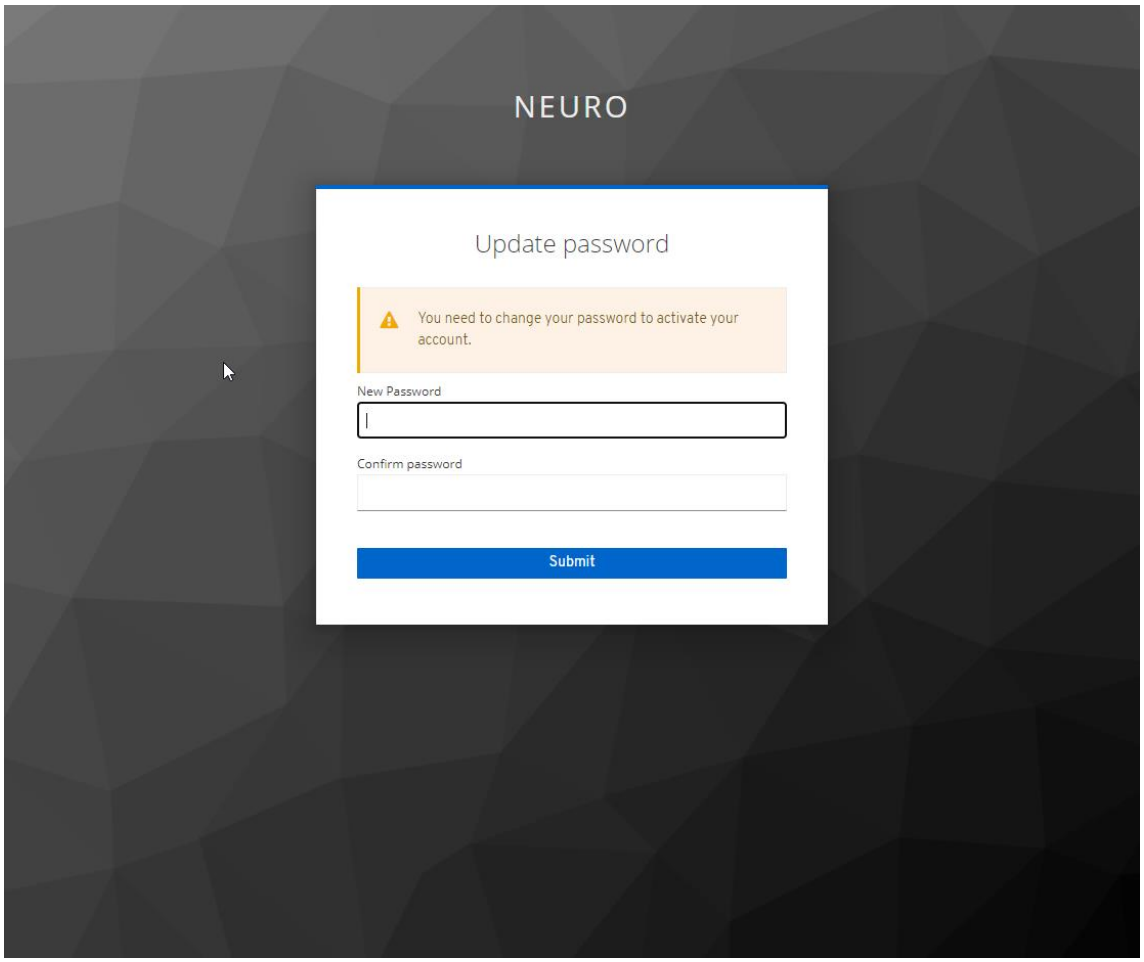
Cuando un nuevo usuario del sistema inicia sesión por primera vez, se le pide que:

- Cambie su contraseña (ya que la actual es generada aleatoriamente y de un solo uso)
- Configure su aplicación TOTP: Por ejemplo Google authenticator

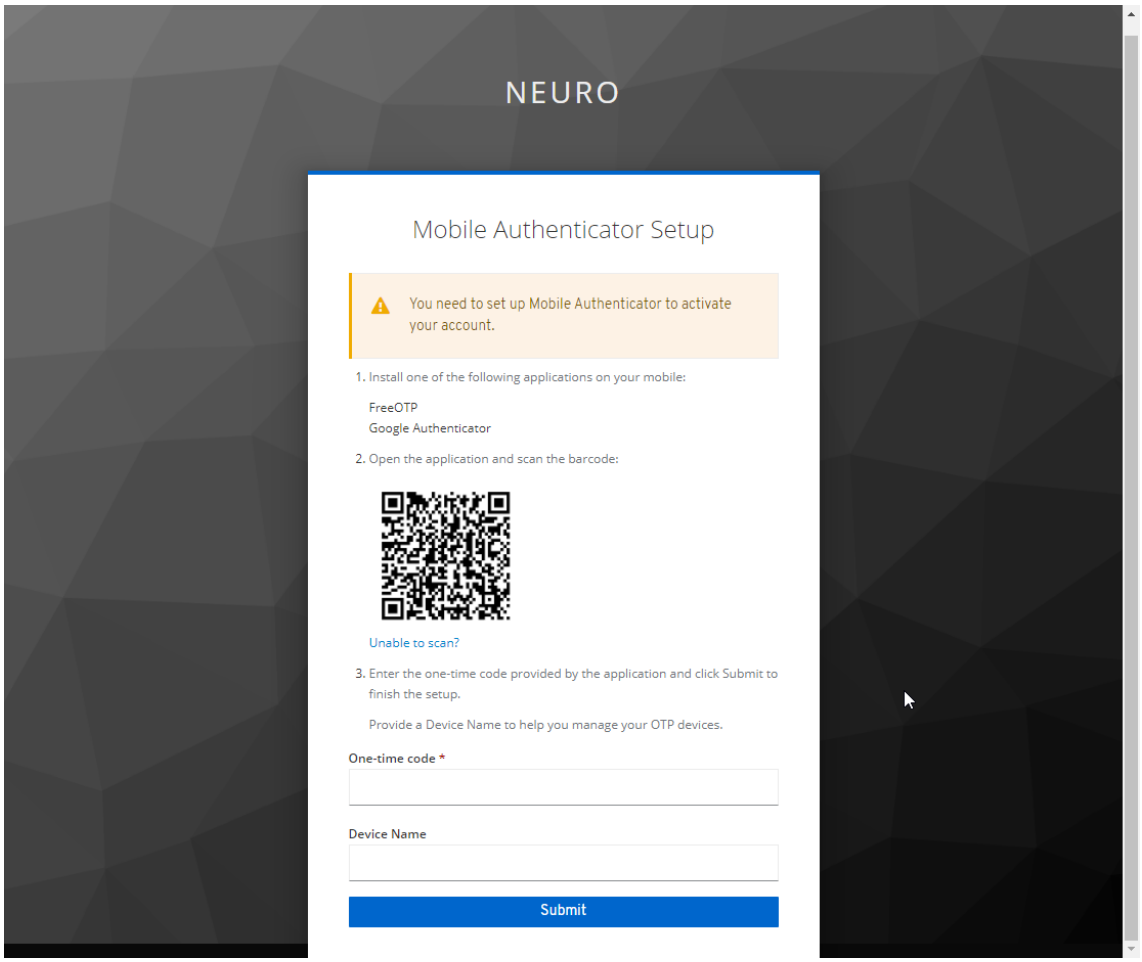
Al acceder a cualquier parte de la app se pedirá login:



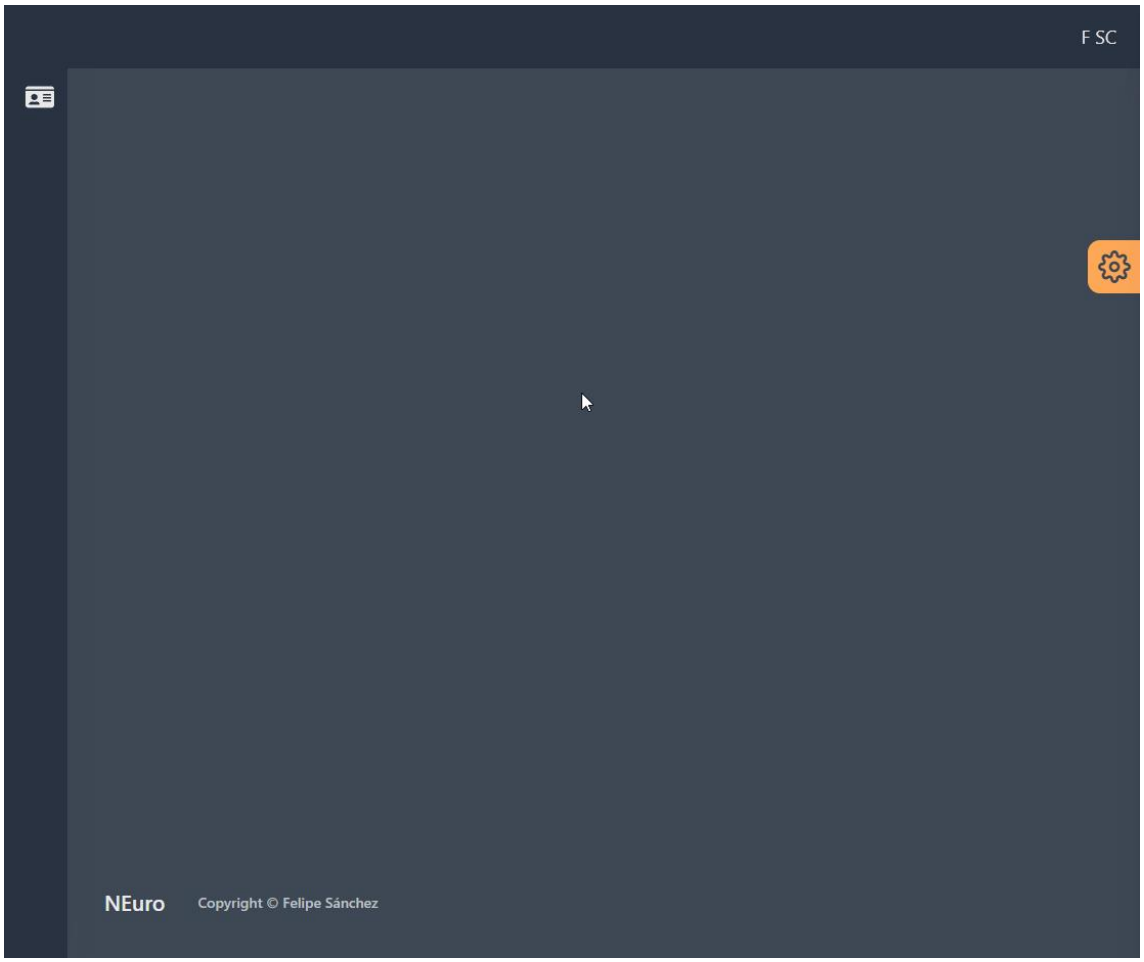
A continuación se solicita el cambio de contraseña:



Después se configura la configuración de TOTP:



Finalmente, habremos iniciado sesión por primera vez:

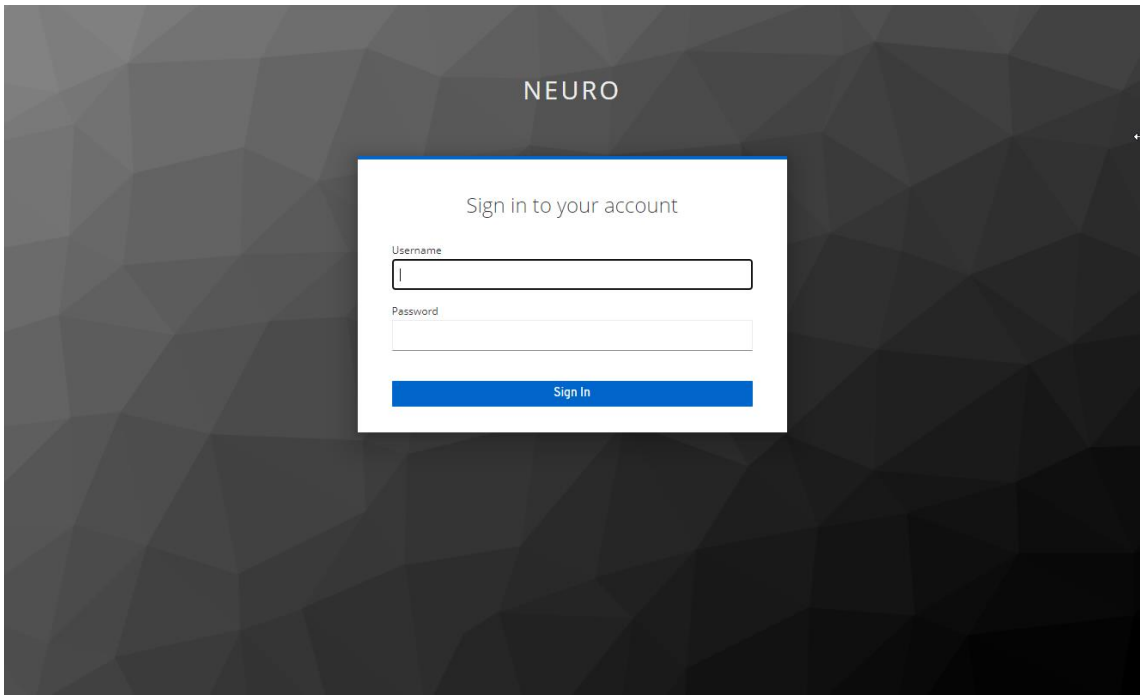


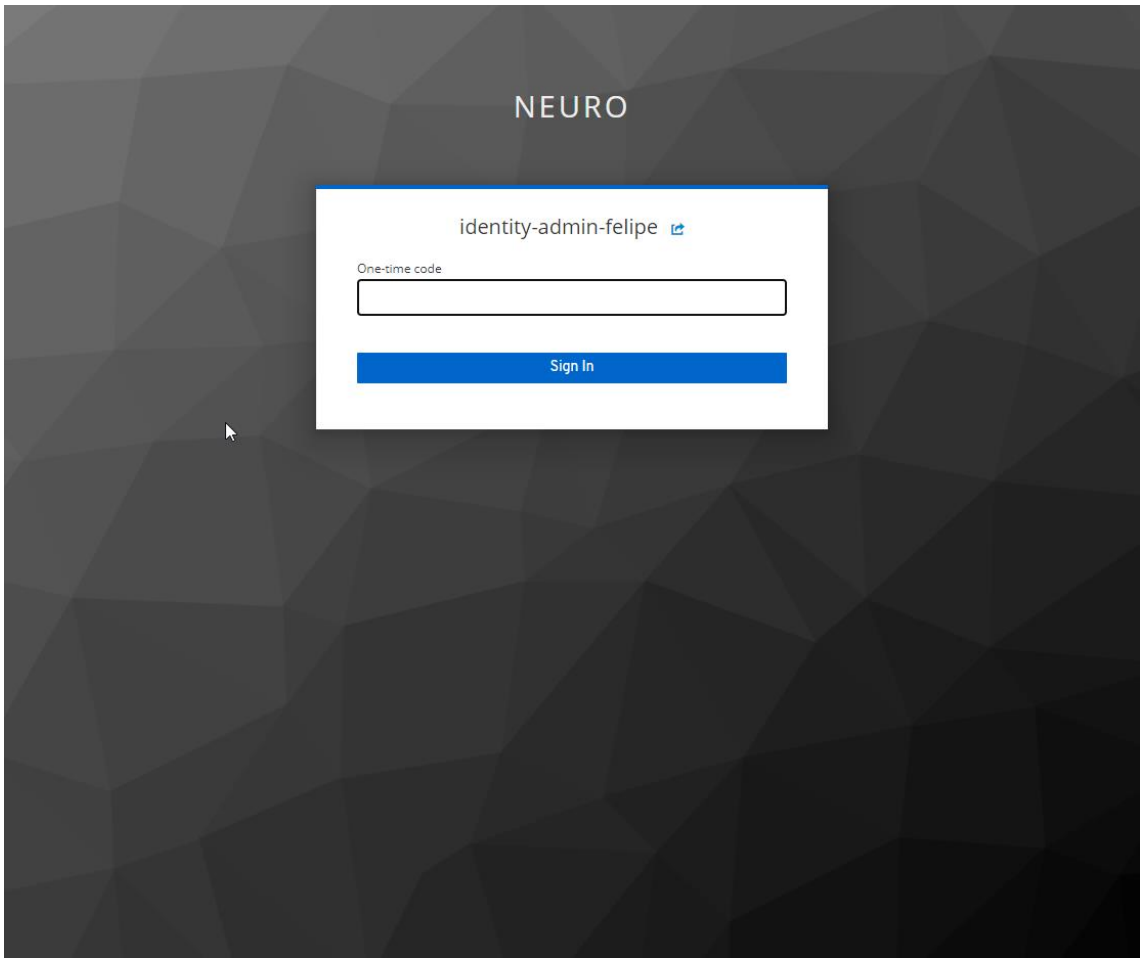
3.1.2 INICIOS DE SESIÓN REGULARES

ROLES: Cualquiera

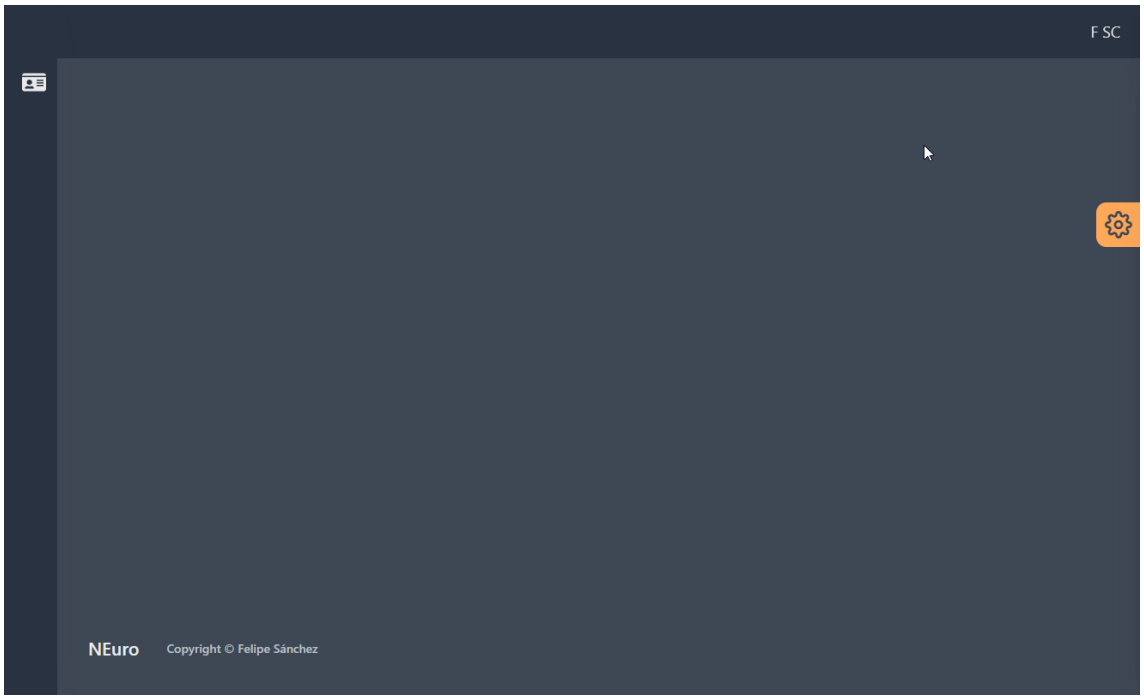
Desde cualquier punto de la aplicación, si no se ha iniciado sesión antes, se redirige al inicio de sesión.

Para iniciar sesión hay que rellenar el formulario con el nombre de usuario y contraseña, y hacer click en “Sign In”. Posteriormente, introducir el código de un solo uso de la aplicación TOTP que se ha elegido.





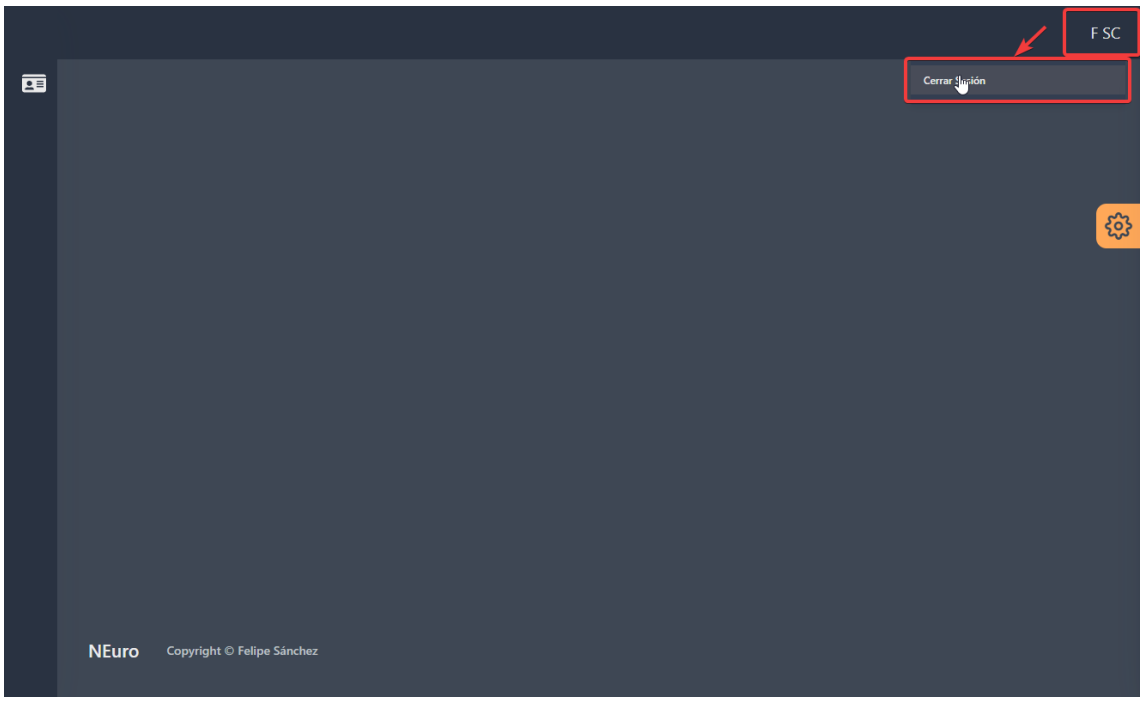
Una vez se ha iniciado sesión, se redirigirá a una pagina de inicio, que no muestra ninguna información. Además, ya se puede ver en la parte superior derecha el nombre de la identidad logada



3.2 CIERRE DE SESIÓN

ROLES: Cualquiera

Para cerrar sesión basta con hacer click en el nombre de la identidad, situado en la parte superior derecha y posteriormente hacer click en “Cerrar Sesion”.

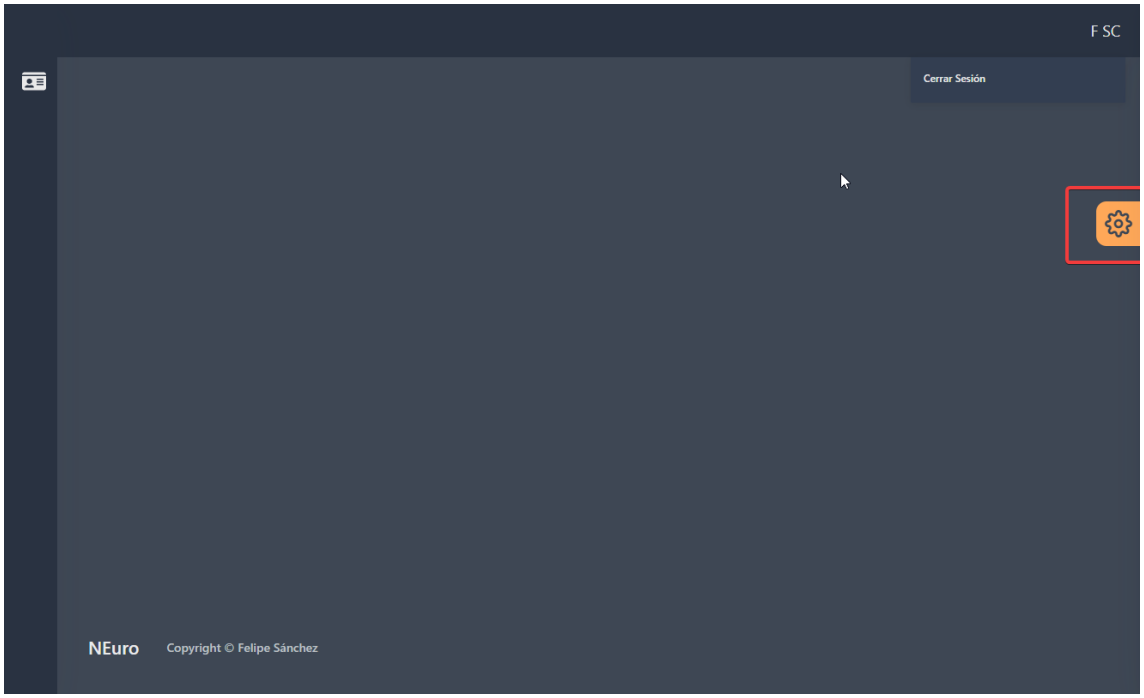


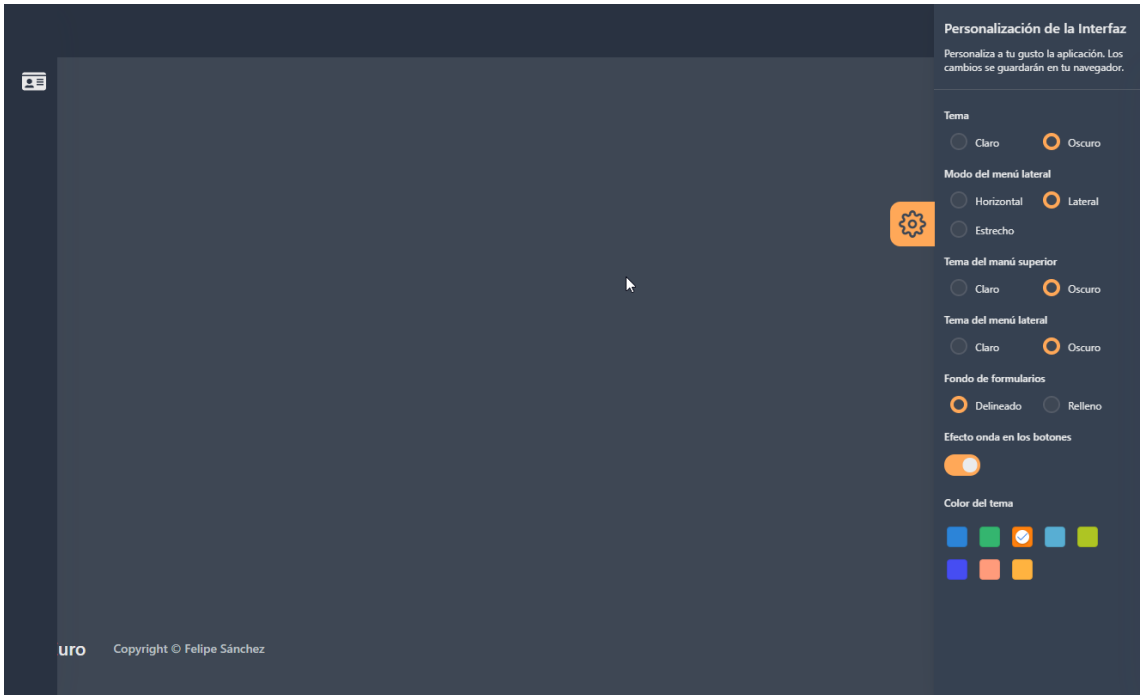
Una vez cerrada la sesión, se redirigirá a la pagina de inicio de sesión.

3.3 CUSTOMIZACIÓN DE LA INTERFAZ

Neuro permite hacer una personalización bastante detallada de la interfaz gráfica. La configuración definida será almacenada en el navegador del usuario y se puede cambiar las veces que se quiera.

Para acceder a la personalización basta con hacer click en el icono de la rueda dentada situado en la parte derecha de la aplicación, e ir haciendo las modificaciones que se quieran.



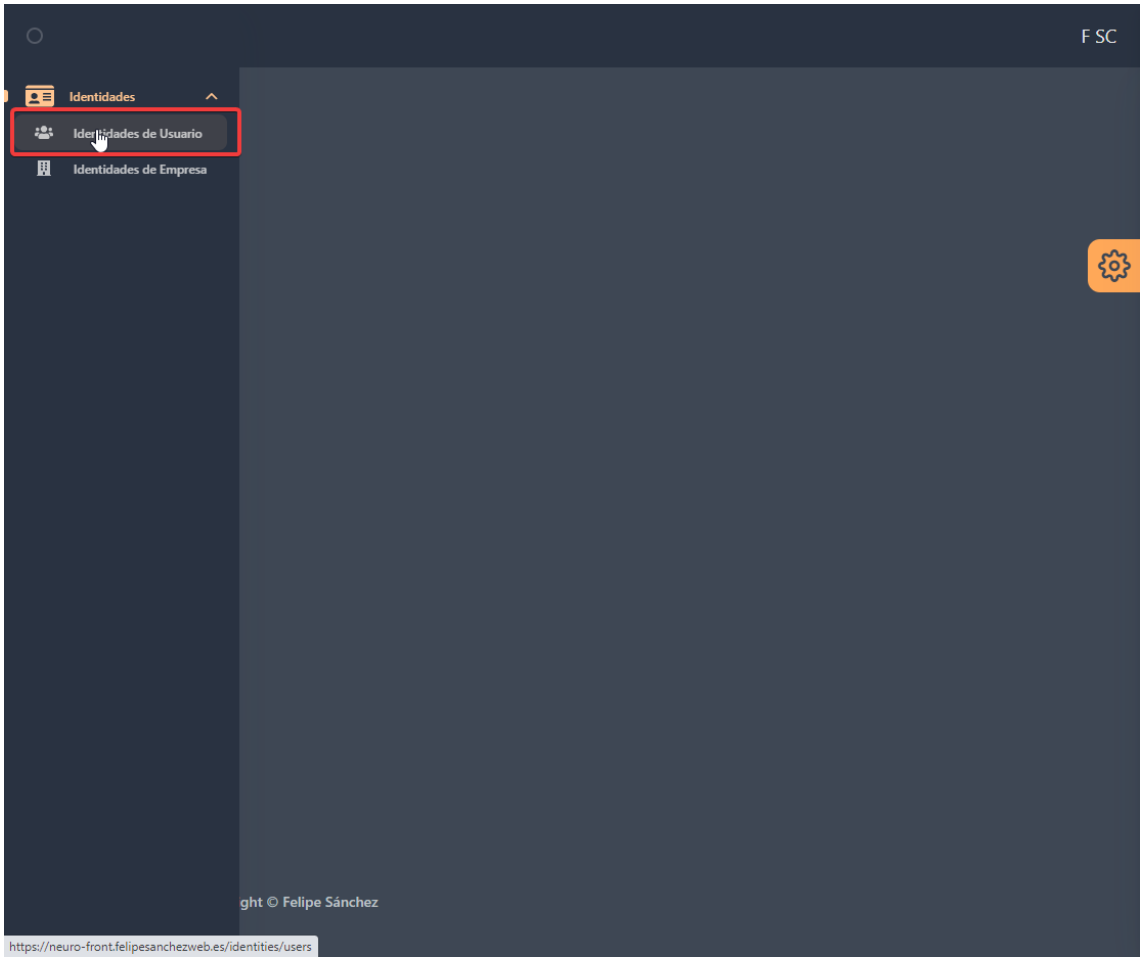


3.4 IDENTIDADES

3.4.1 GENERACIÓN DE IDENTIDADES DE USUARIO

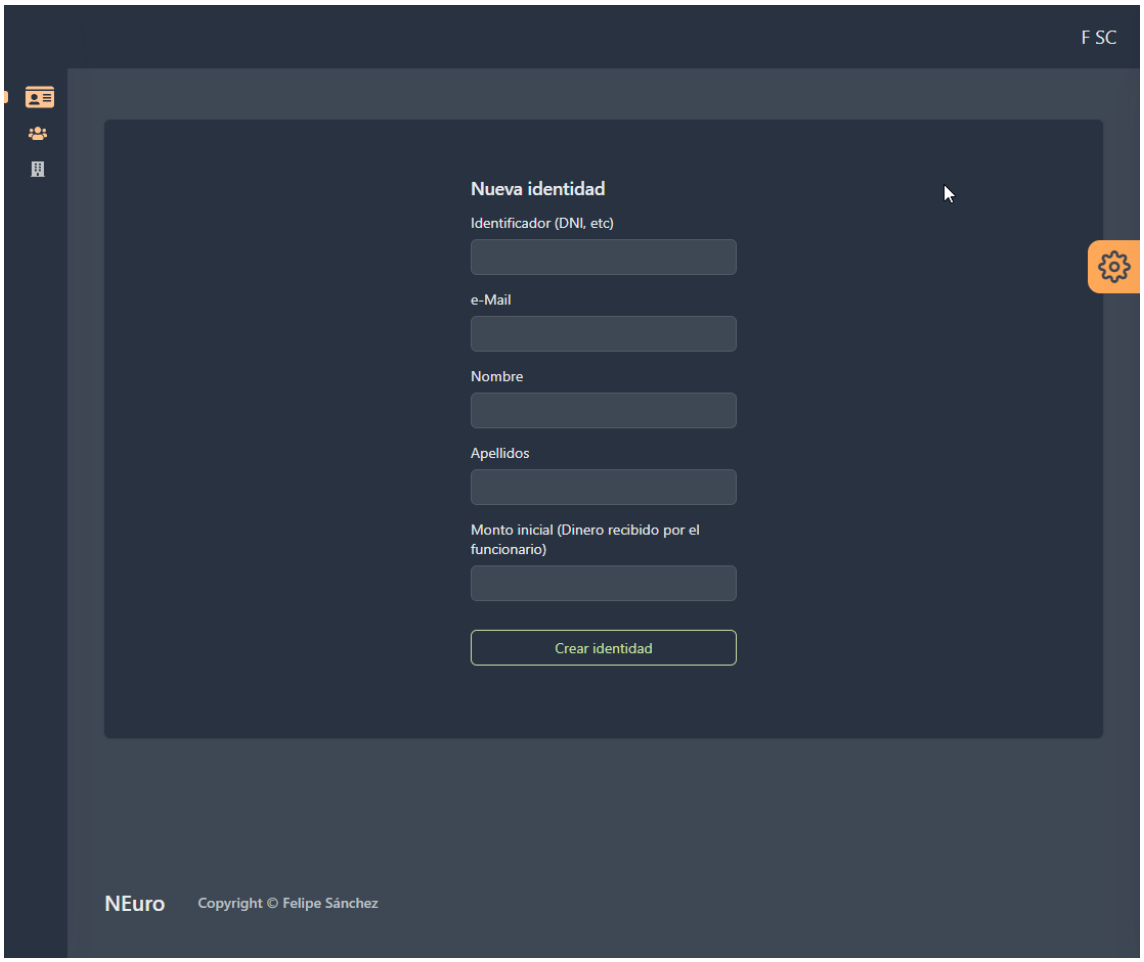
ROLES: Administrador de identidades

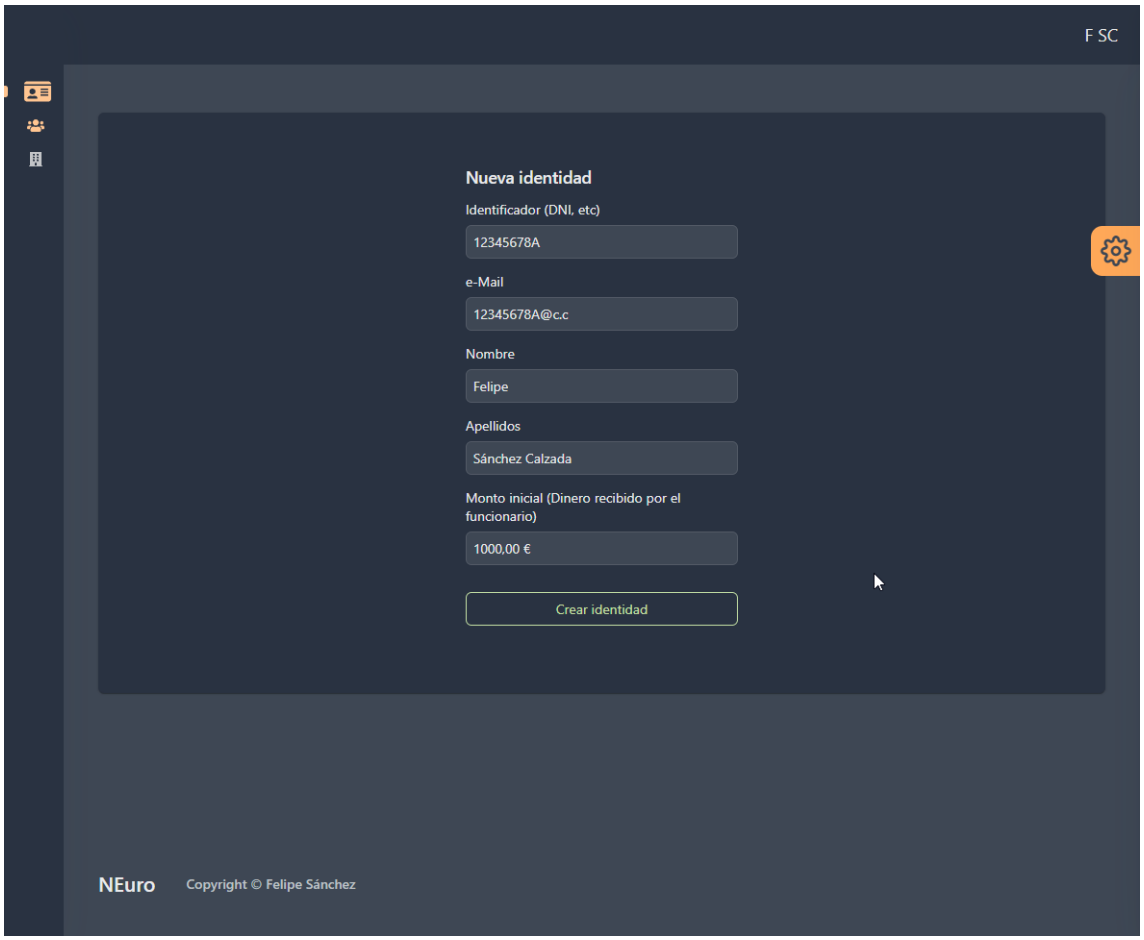
Para ello hay que dirigirse a: Identidades -> Identidades de Usuario



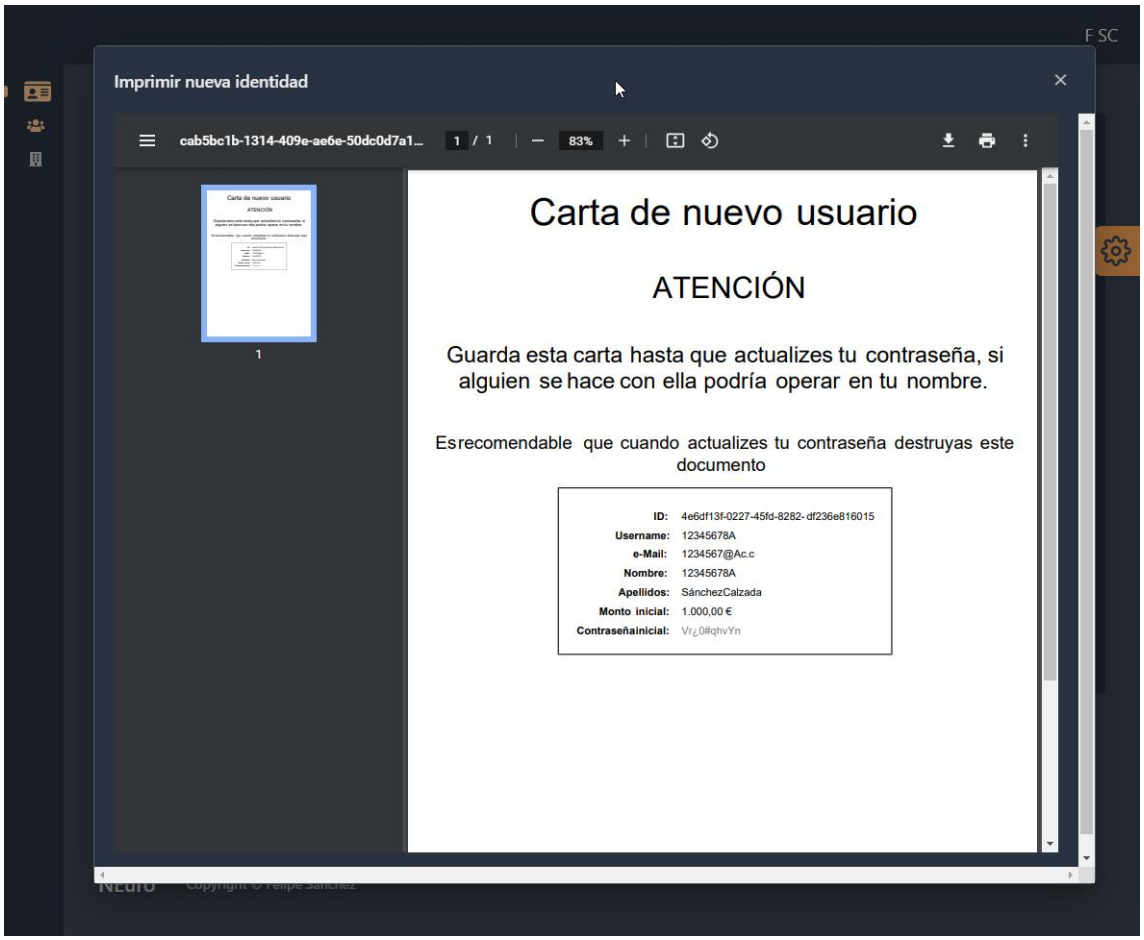
Después hay que rellenar el formulario con los datos del nuevo usuario.

EN este punto es donde el funcionario ha ce hacer la recogida del dinero con el que se inicializar la cuenta bancaria por defecto que se va a generar junto con el nuevo usuario.


F SC
Nueva identidad
Identificador (DNI, etc)
e-Mail
Nombre
Apellidos
Monto inicial (Dinero recibido por el funcionario)
Crear identidad
NEuro
Copyright © Felipe Sánchez



Una vez relleno el formulario, y hacendo click en Crear identidad, se nos va a abrir una nueva ventana con el PDF que contiene la información de la nueva cuenta y su primera contraseña.



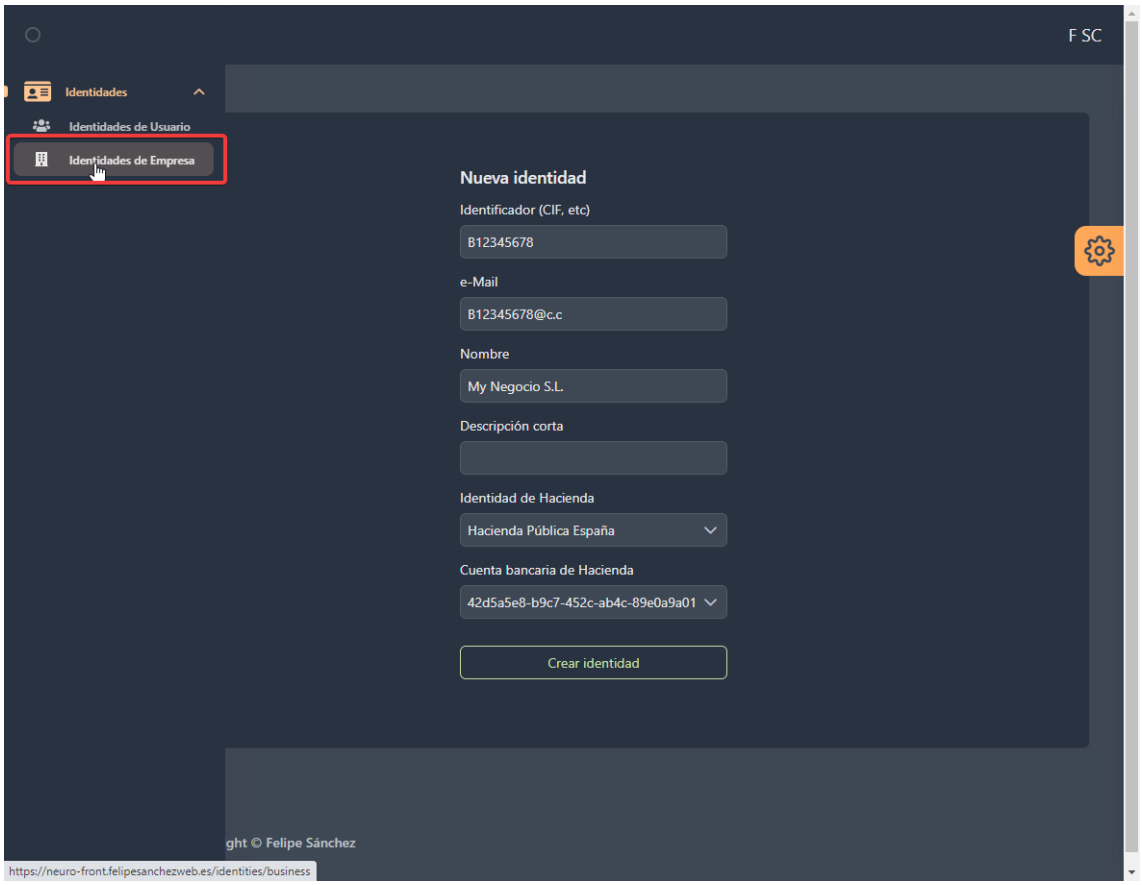
El funcionario solo tendría que imprimir la carta y entregársela a la persona que solicitó el alta en el sistema.

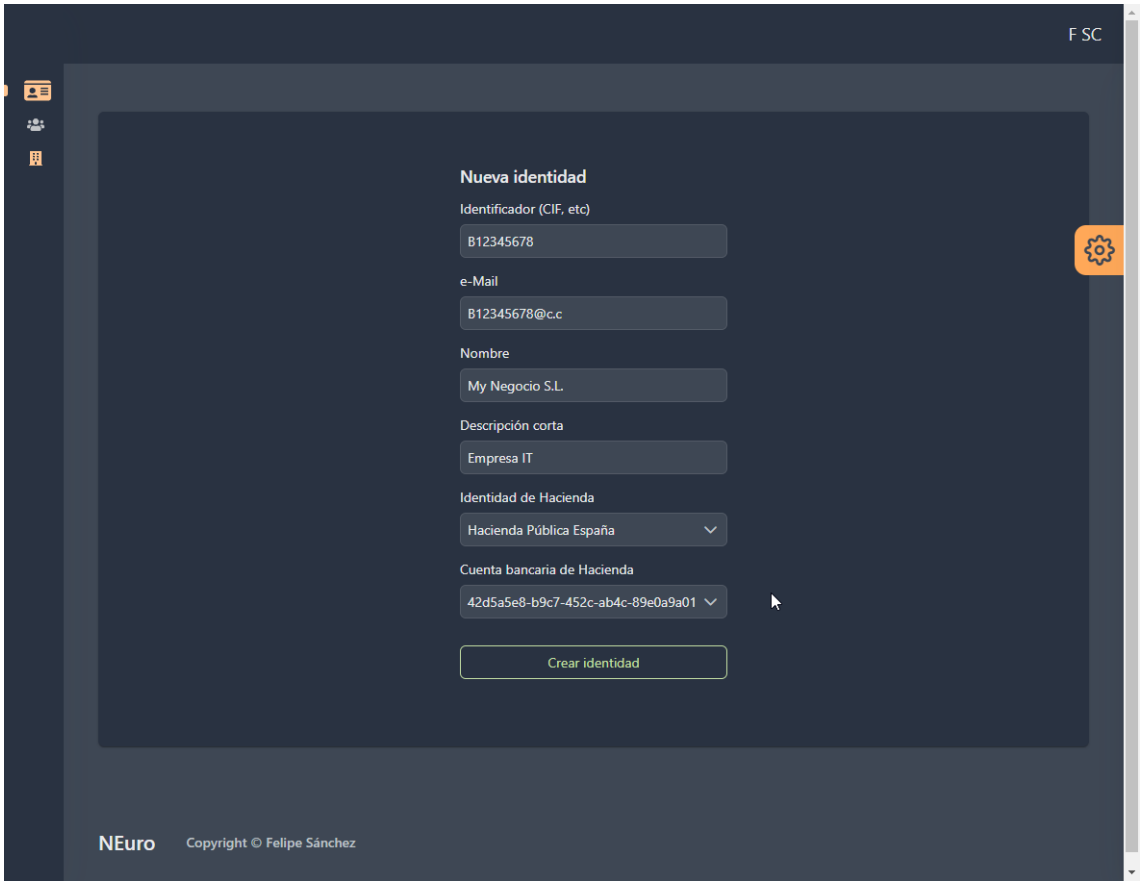
3.4.2 GENERACIÓN DE IDENTIDADES DE EMPRESA

ROLES: Administrador de identidades

Una empresa hace las mismas funciones que un usuario, pero además puede expedir facturas (y sus impuestos se pagará automaticamente).

Para ello, el administrador de identidades ha de ir a Identidades -> Identidades de Empresa, y rellenar el formulario.





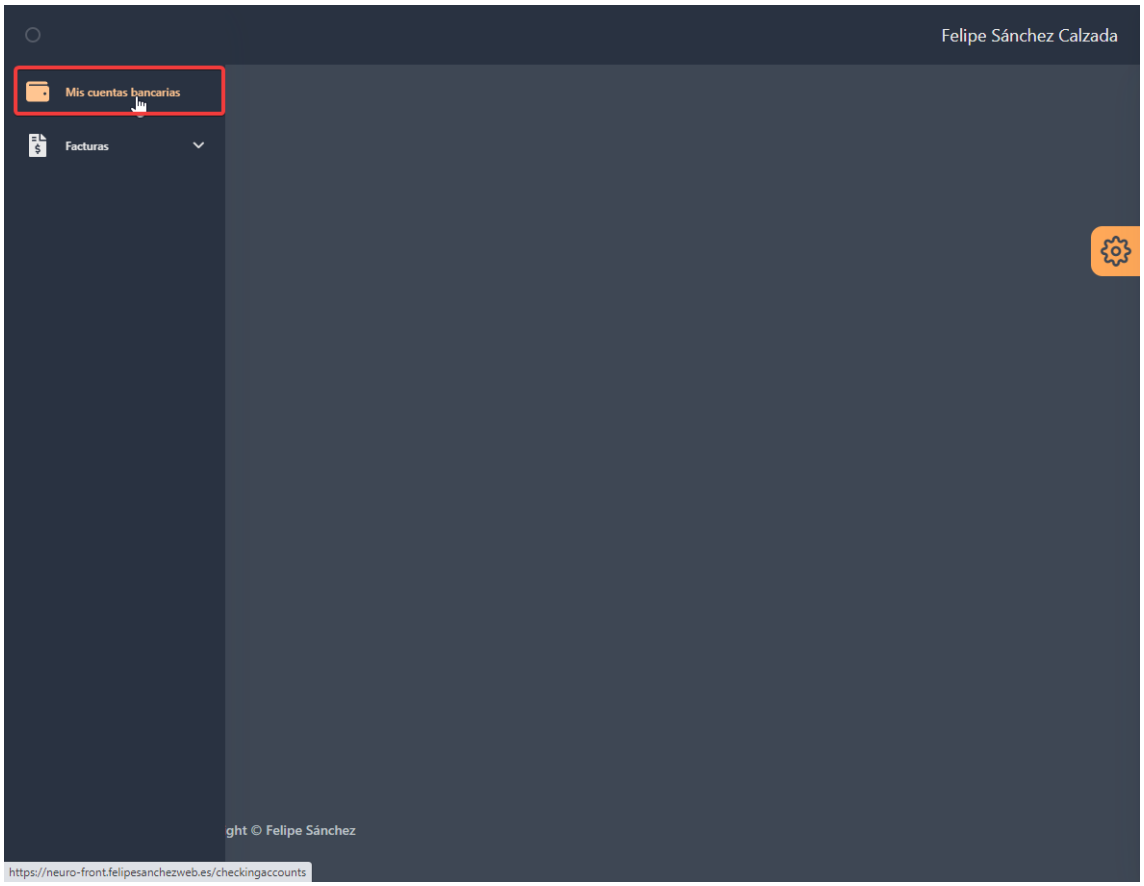
Finalmente, obtenemos una carta de empresa similar a la de los usuarios.

3.5 OPERACIONES CON CUENTAS BANCARIAS

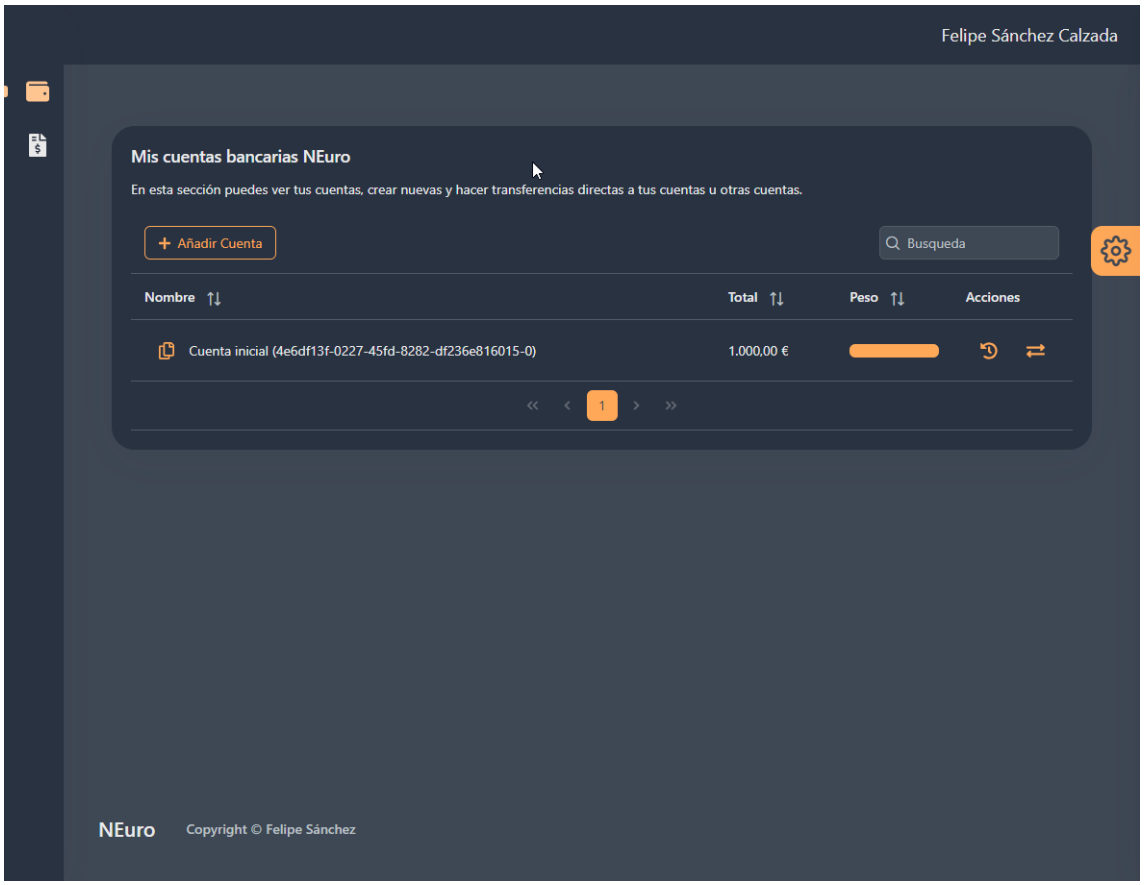
3.5.1 VER MIS CUENTAS BANCARIAS

ROLES: Usuario, Empresa, Hacienda Publica

Para ver las cuentas bancarias basta con dirigirse a “Mis cuentas bancarias”



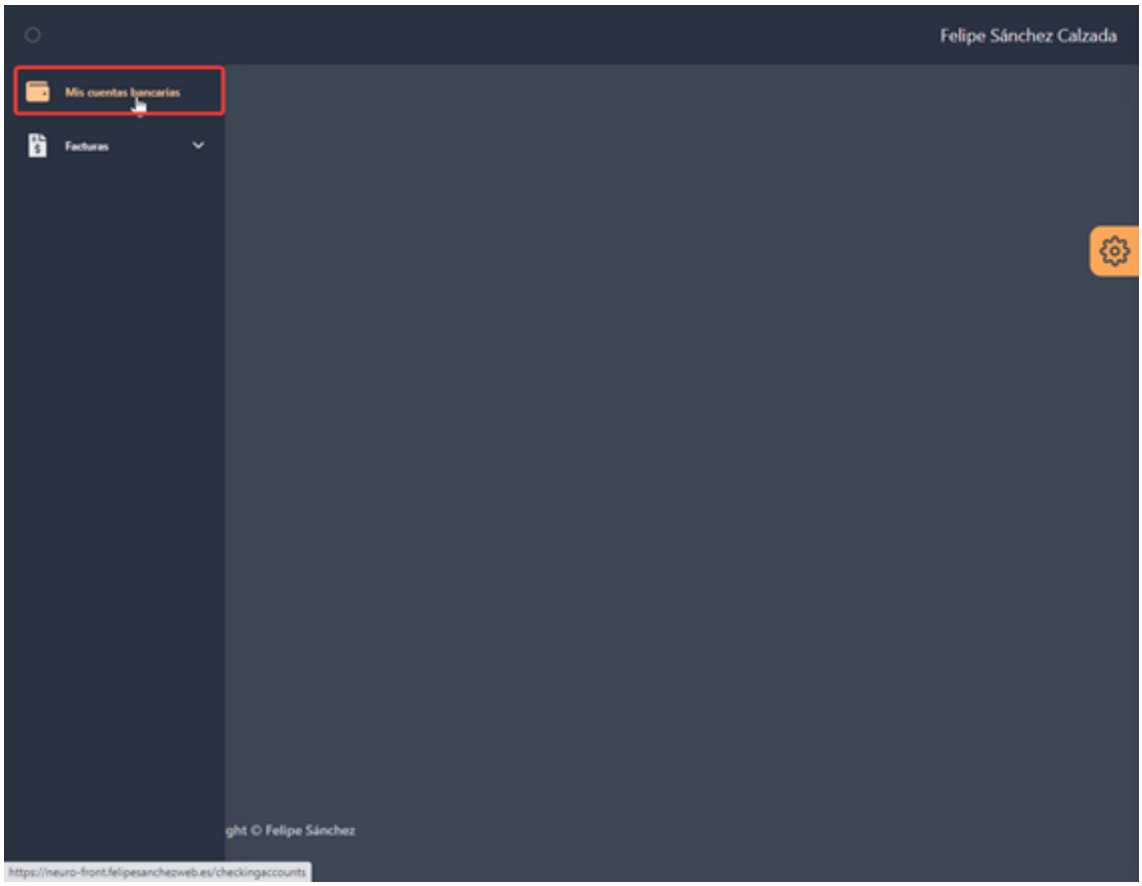
Se mostrará una tabla con todas las cuentas bancarias, su información y el peso de la cuenta. El peso es el porcentaje de dinero que hay en ese cuenta con respecto a la suma del dinero de todas las cuentas.

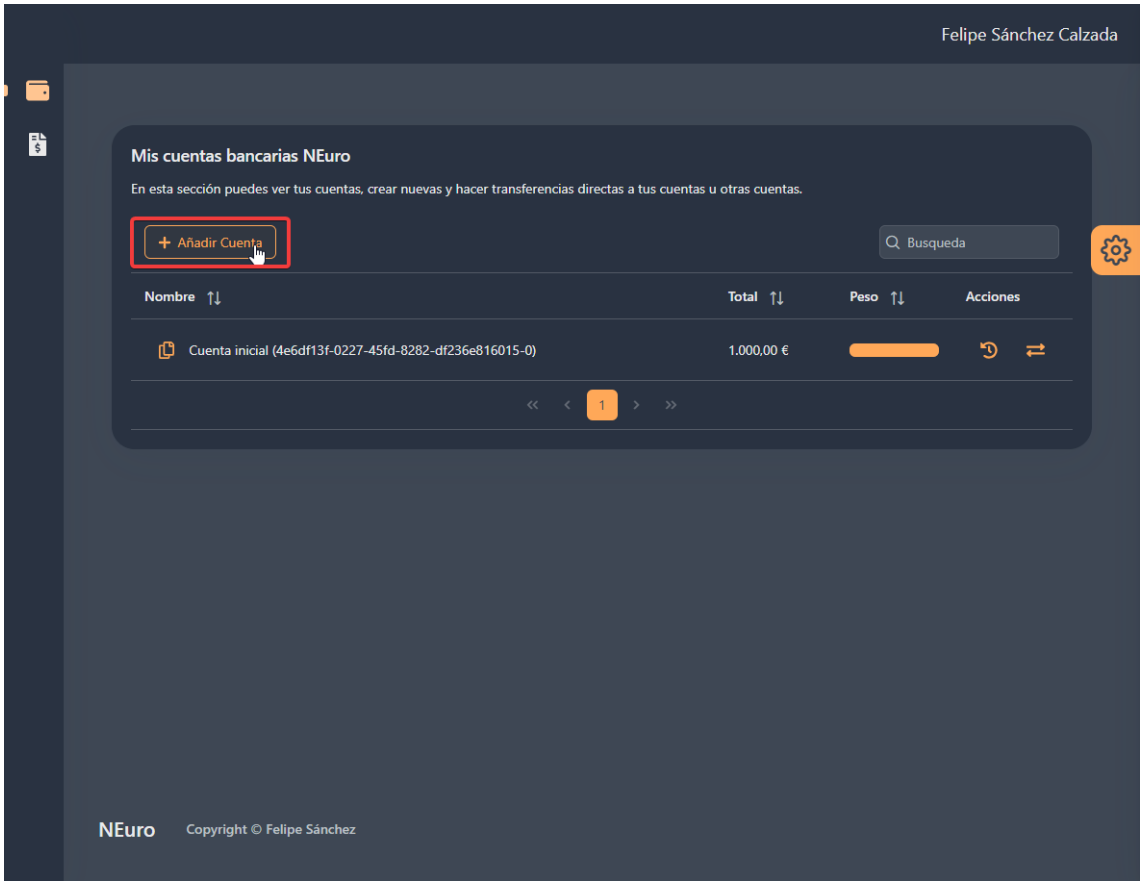


3.5.2 CREAR NUEVA CUENTA BANCARIA

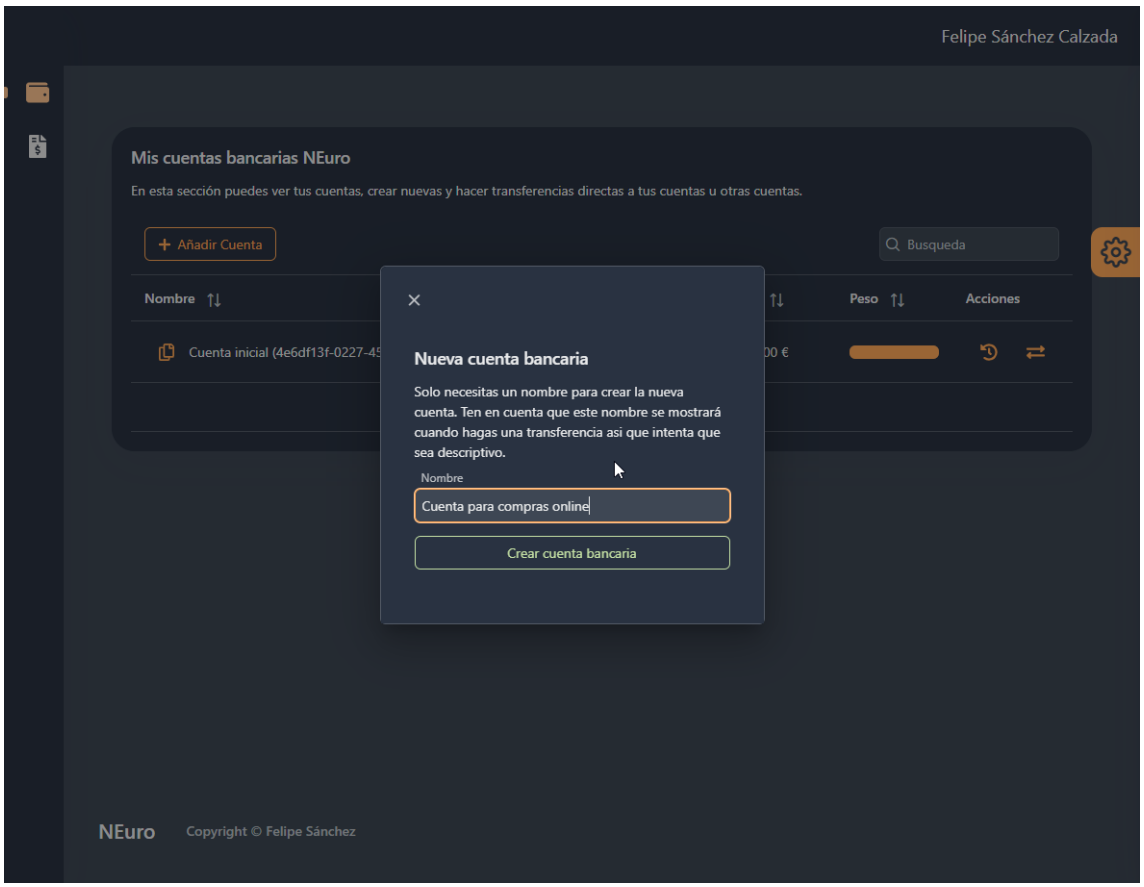
ROLES: Usuario, Empresa, Hacienda Publica

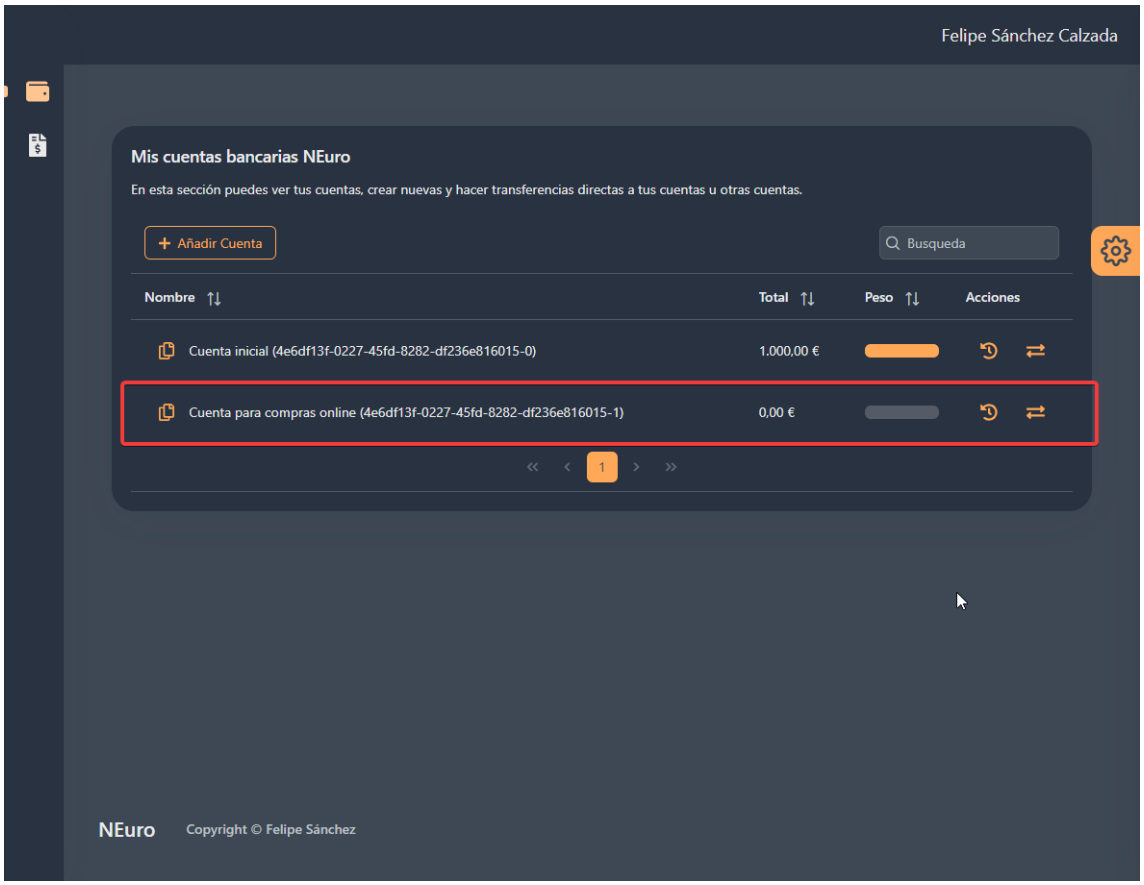
Hay que dirigirse a: “Mis cuentas bancarias” y después hacer click en “Añadir Cuenta”


Felipe Sánchez Calzada
Mis cuentas bancarias
Facturas
ght © Felipe Sánchez
https://neuro-front.felipesanchezweb.es/checkingaccounts


Felipe Sánchez Calzada
Mis cuentas bancarias NEuro
En esta sección puedes ver tus cuentas, crear nuevas y hacer transferencias directas a tus cuentas u otras cuentas.
+ Añadir Cuenta
Busqueda
Nombre
Total
Peso
Acciones
Cuenta inicial (4e6df13f-0227-45fd-8282-df236e816015-0)
1.000,00 €
1
NEuro
Copyright © Felipe Sánchez

Se abrirá un cuadro de dialogo pidiendo el nombre de la cuenta, basta con rellenarlo y hacer click en “Crear cuenta bancaria”



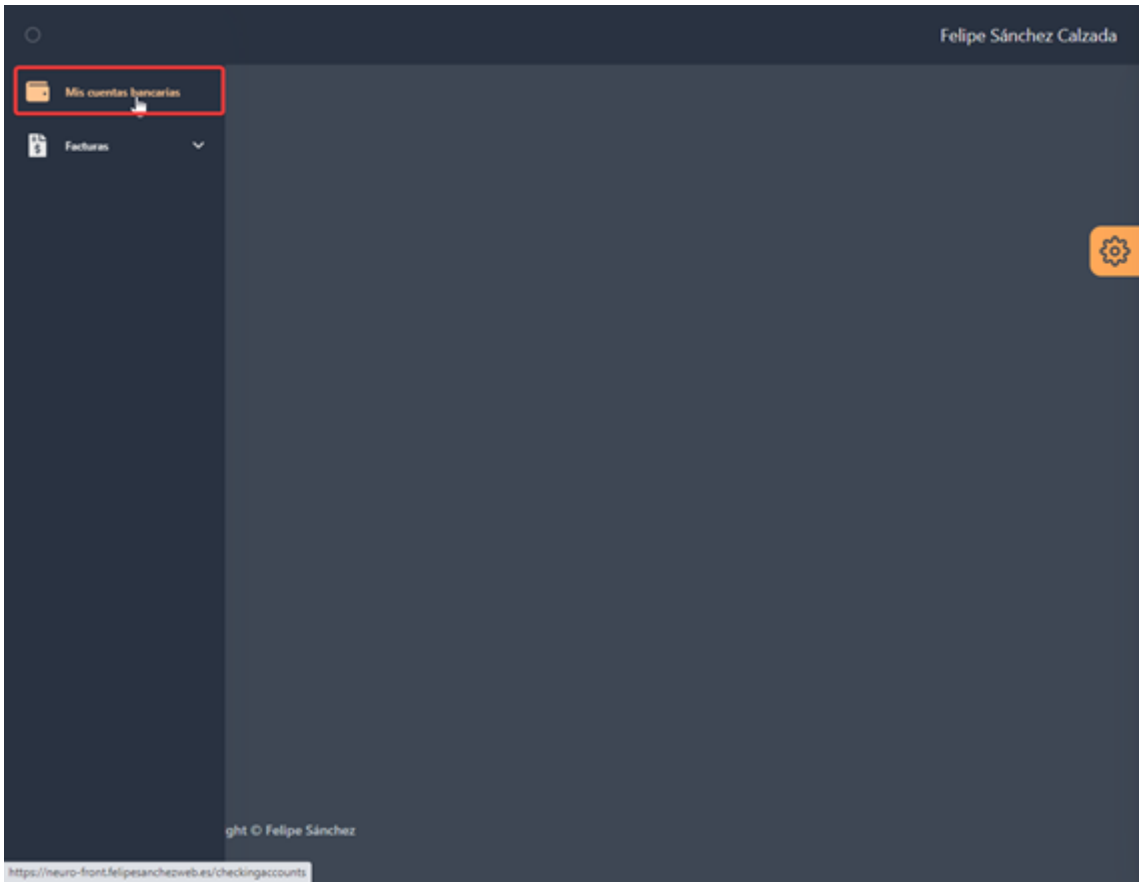


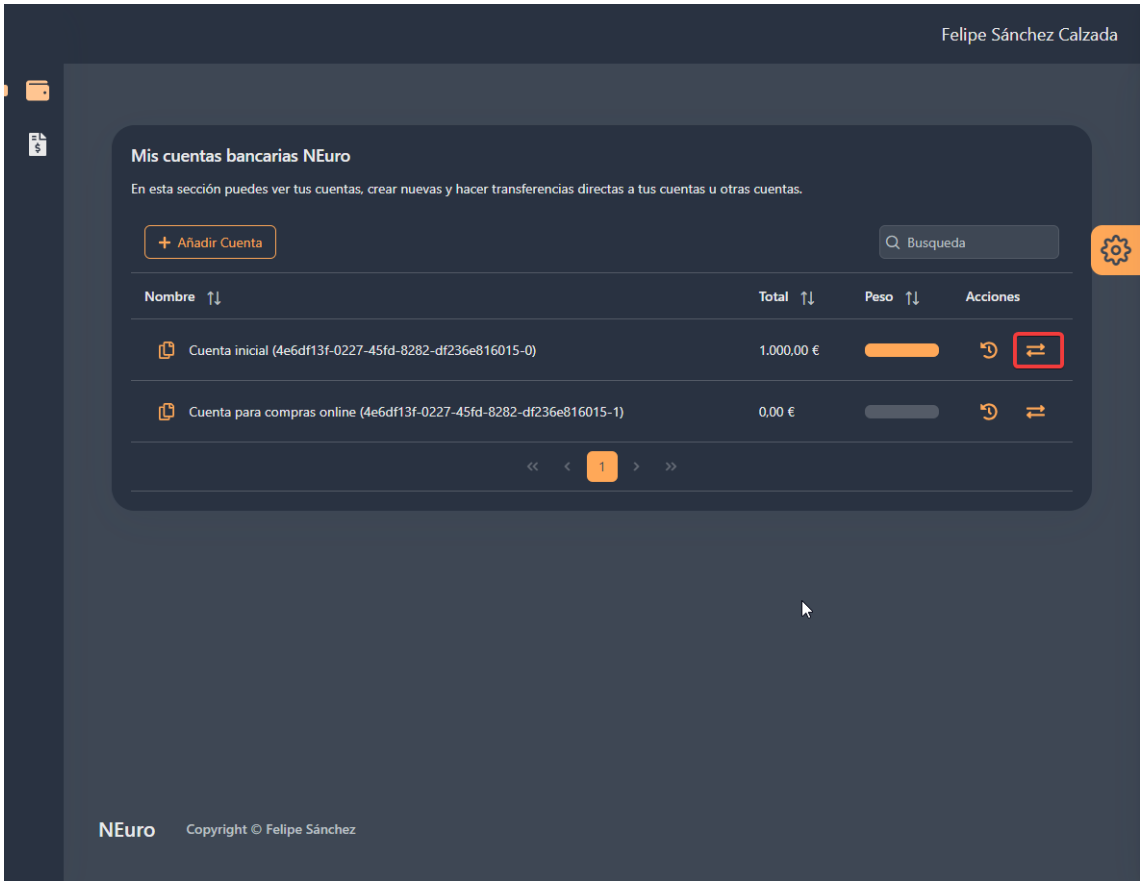
3.5.3 REALIZAR TRANSFERENCIA BANCARIA

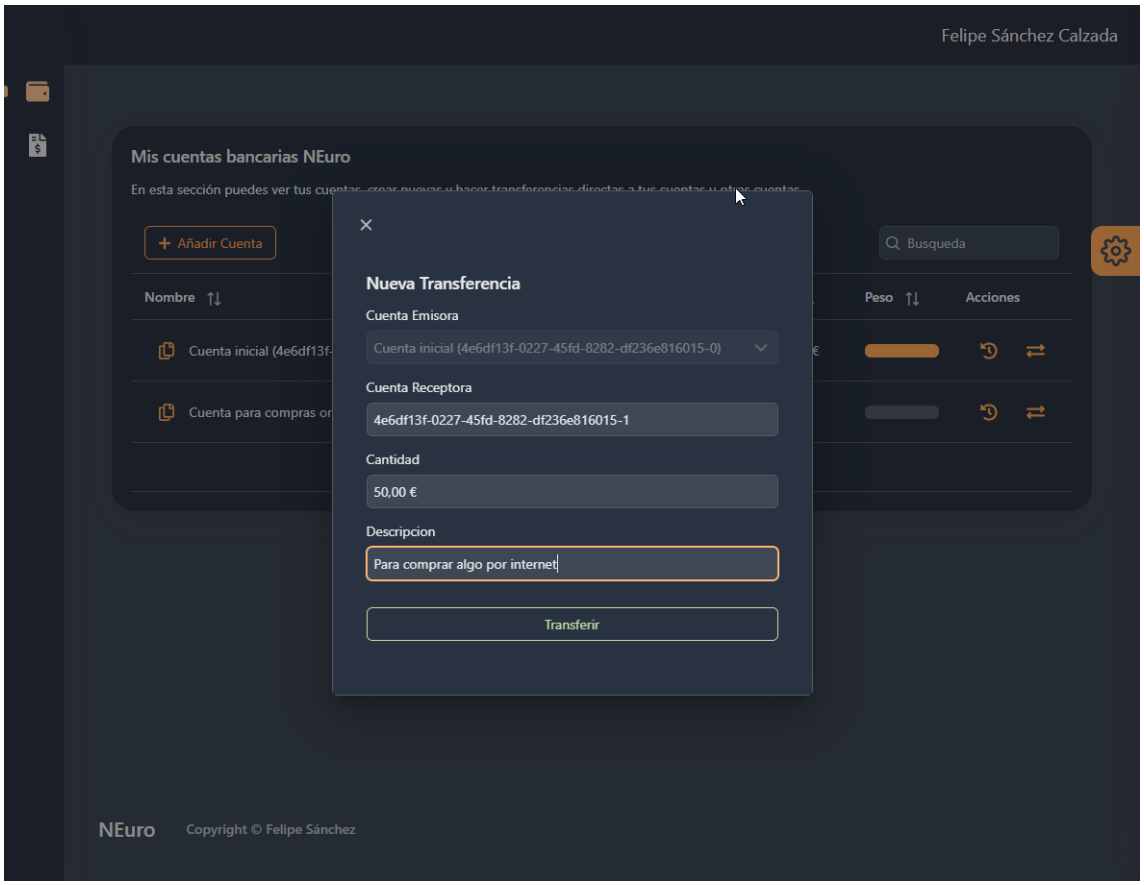
ROLES: Usuario, Empresa, Hacienda Publica

Para hacer una transferencia se necesita el ID de la cuenta destino. Los ID de cuentas bancarias tienen el siguiente formato: 4e6df13f-0227-45fd-8282-df236e816015-1 que se traduce en un UUID junto con un guion y un numero final.

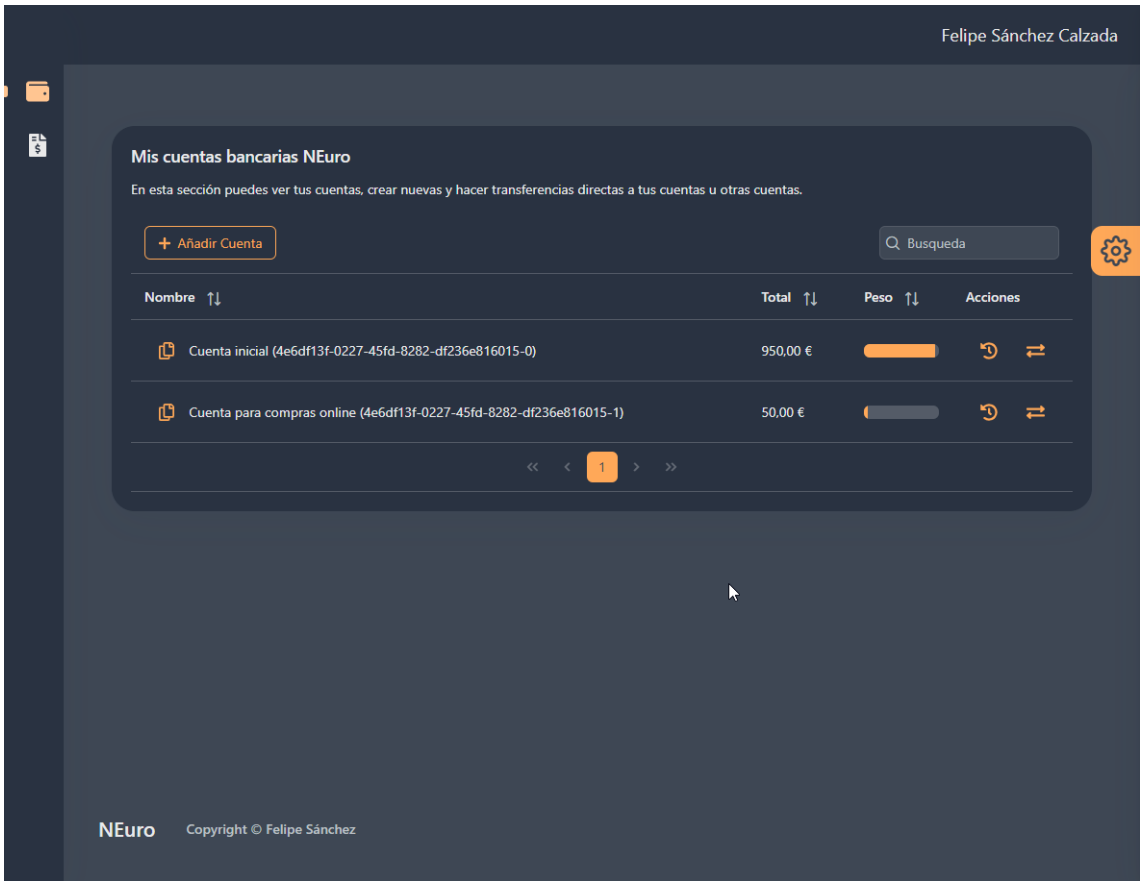
Hay que dirigirse a: "Mis cuentas bancarias" y después hacer click en el icono ⇄ de la tabla de cuentas bancarias:


Felipe Sánchez Calzada
Mis cuentas bancarias
Facturas
ght © Felipe Sánchez
https://neuro-front.felipesanchezweb.es/checkingaccounts


Felipe Sánchez Calzada
Mis cuentas bancarias NEuro
En esta sección puedes ver tus cuentas, crear nuevas y hacer transferencias directas a tus cuentas u otras cuentas.
+ Añadir Cuenta
Busqueda
Nombre
Total
Peso
Acciones
Cuenta inicial (4e6df13f-0227-45fd-8282-df236e816015-0)
1.000,00 €
Cuenta para compras online (4e6df13f-0227-45fd-8282-df236e816015-1)
0,00 €
1
NEuro
Copyright © Felipe Sánchez



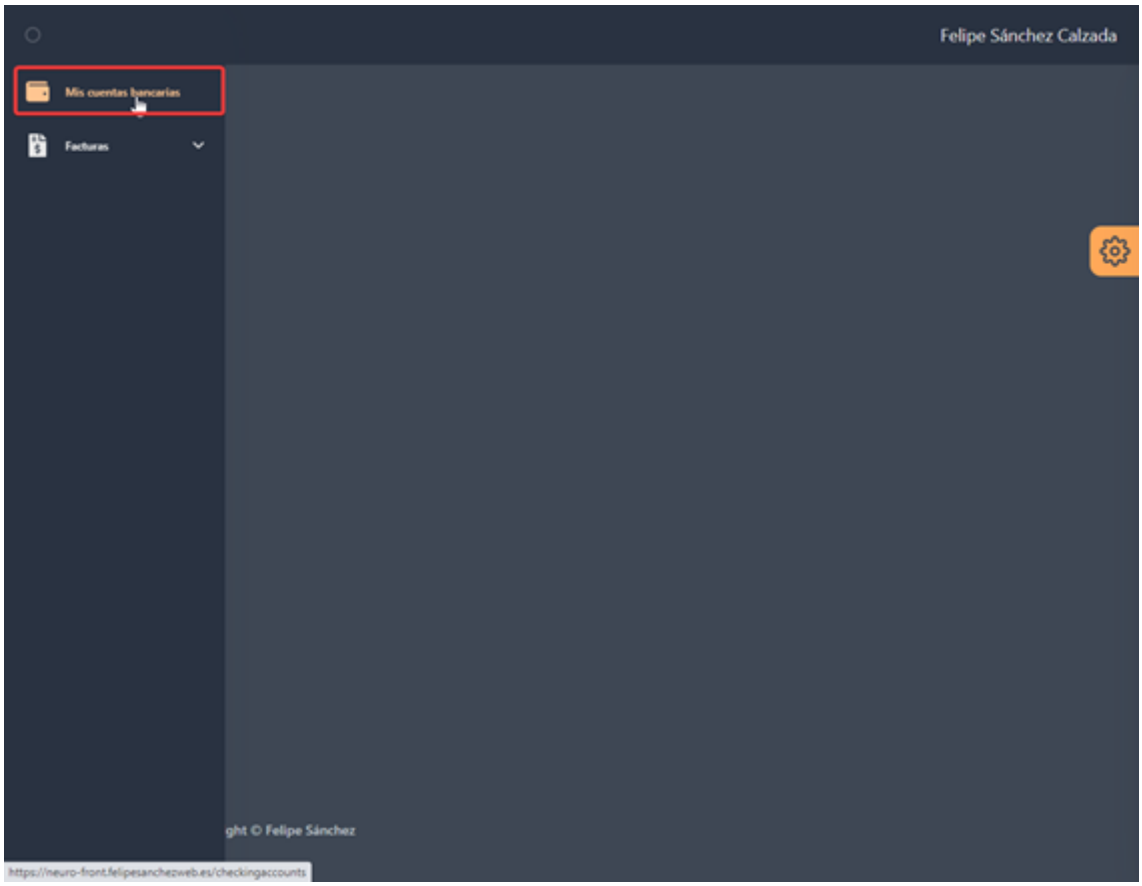
Para la muestra se ha hecho una transferencia a una cuenta del propio usuario que envía el dinero:

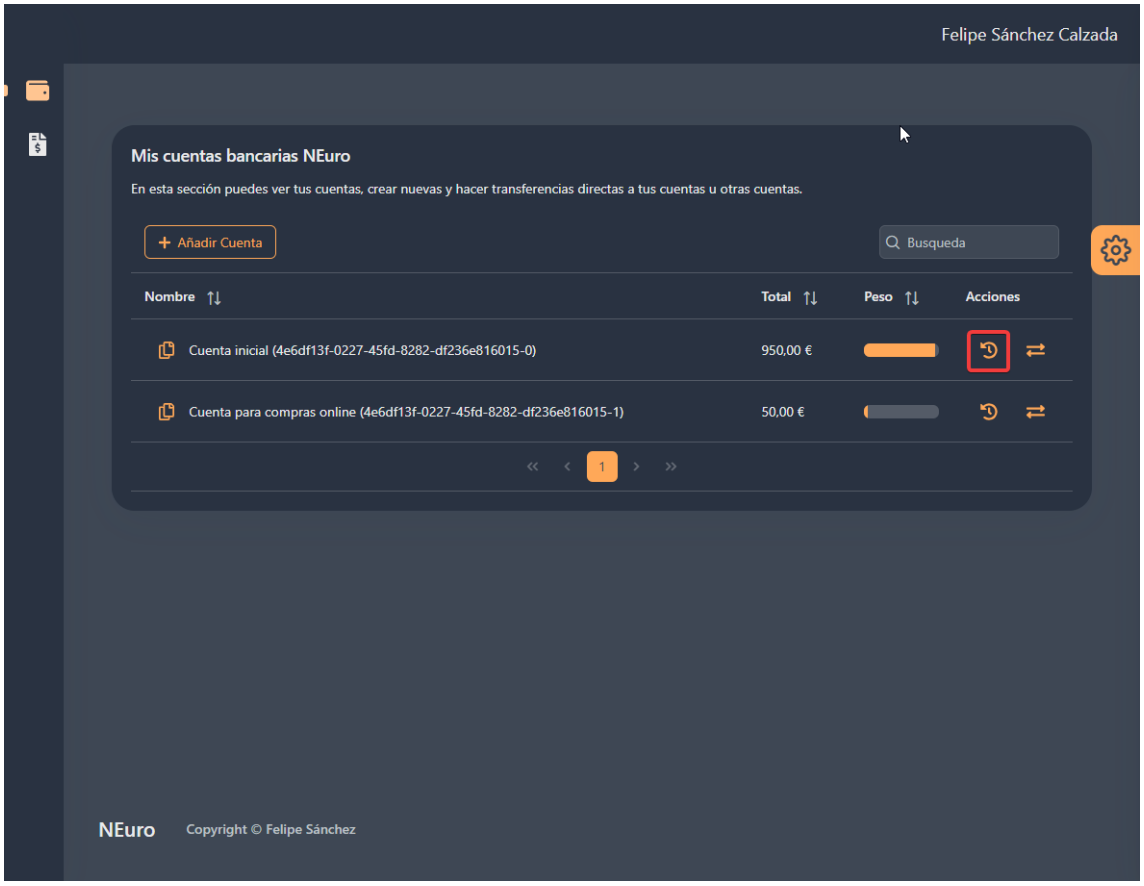


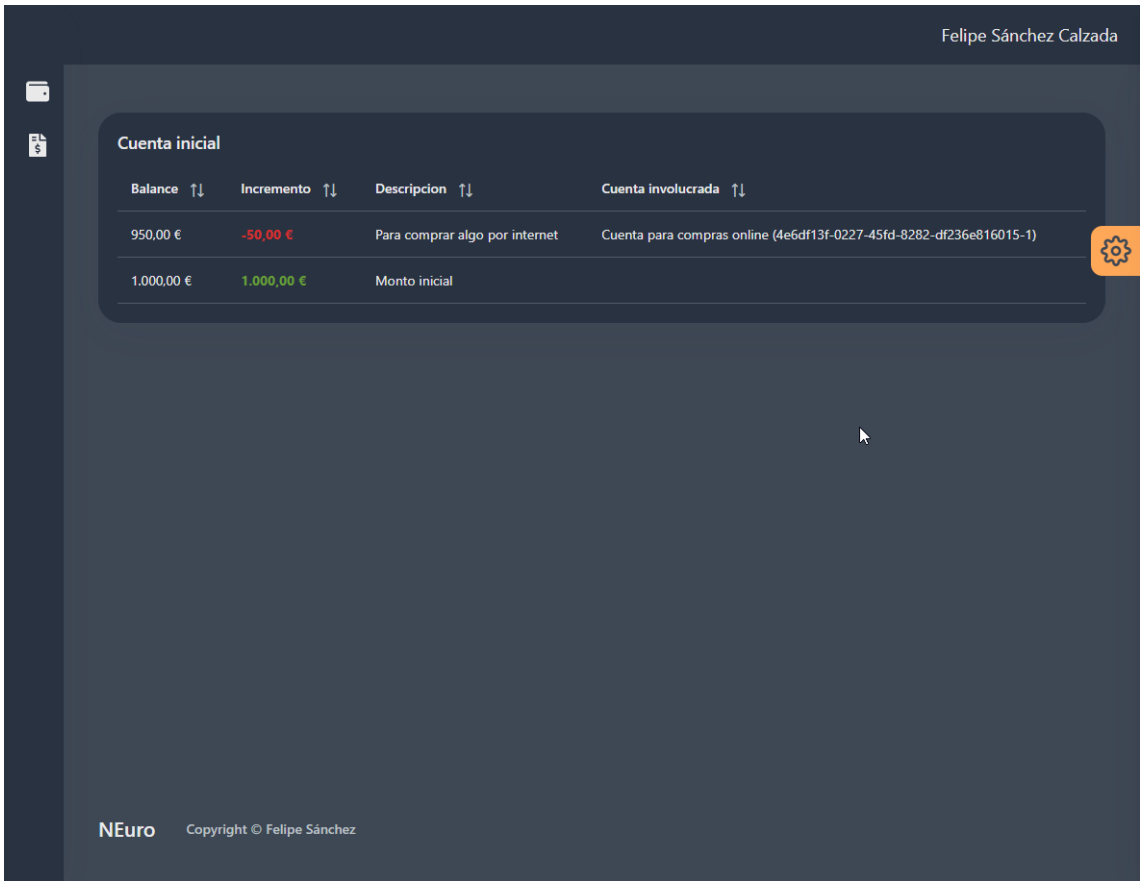
3.5.4 VISUALIZACIÓN DE LOS MOVIMIENTOS DE UNA CUENTA

ROLES: Usuario, Empresa, Hacienda Publica

Hay que dirigirse a: Mis cuentas bancarias y hacer click en el icono 


Felipe Sánchez Calzada
Mis cuentas bancarias
Facturas
ght © Felipe Sánchez
https://neuro-front.felipesanchezweb.es/checkingaccounts


Felipe Sánchez Calzada
Mis cuentas bancarias NEuro
En esta sección puedes ver tus cuentas, crear nuevas y hacer transferencias directas a tus cuentas u otras cuentas.
+ Añadir Cuenta
Busqueda
Nombre
Total
Peso
Acciones
Cuenta inicial (4e6df13f-0227-45fd-8282-df236e816015-0)
950,00 €
Cuenta para compras online (4e6df13f-0227-45fd-8282-df236e816015-1)
50,00 €
1
NEuro
Copyright © Felipe Sánchez

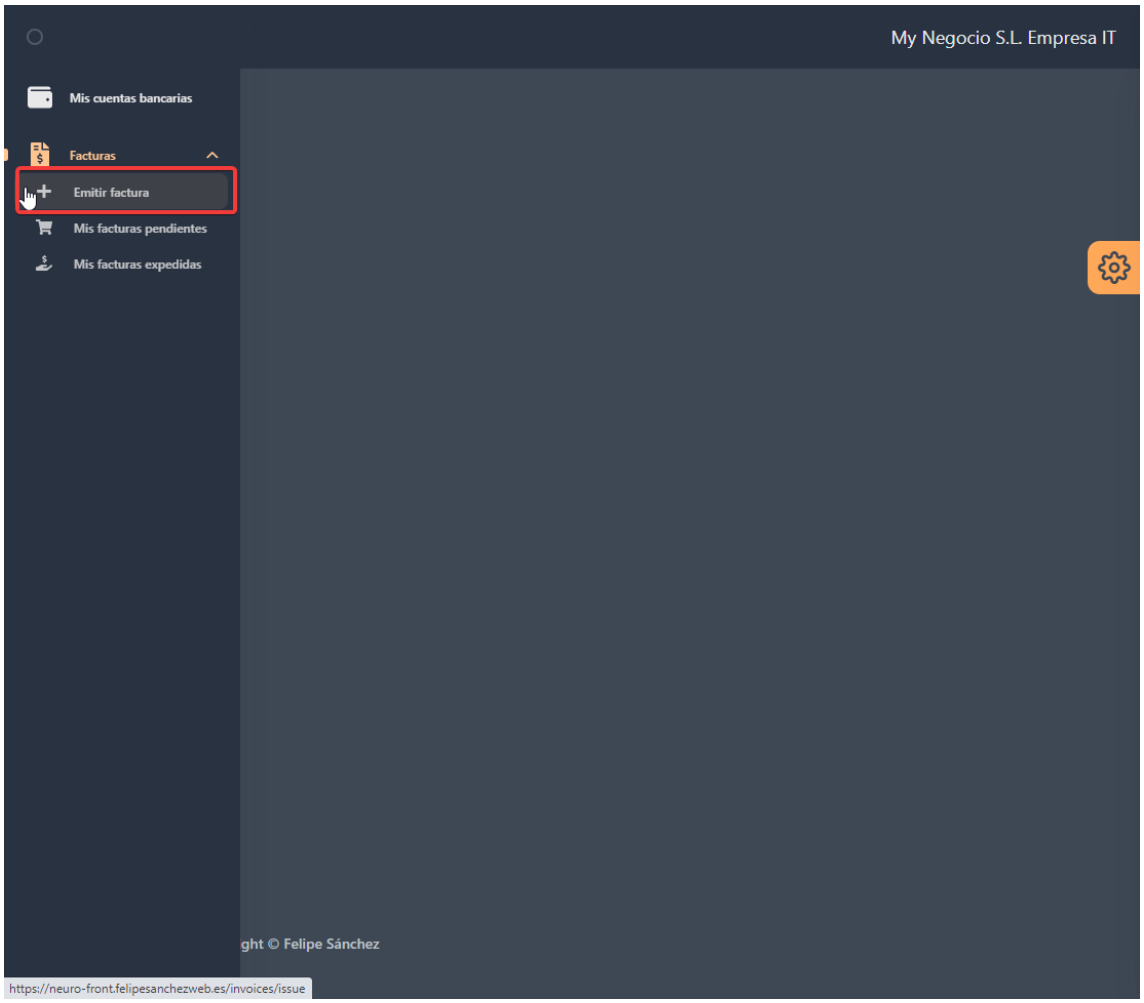


3.6 FACTURAS

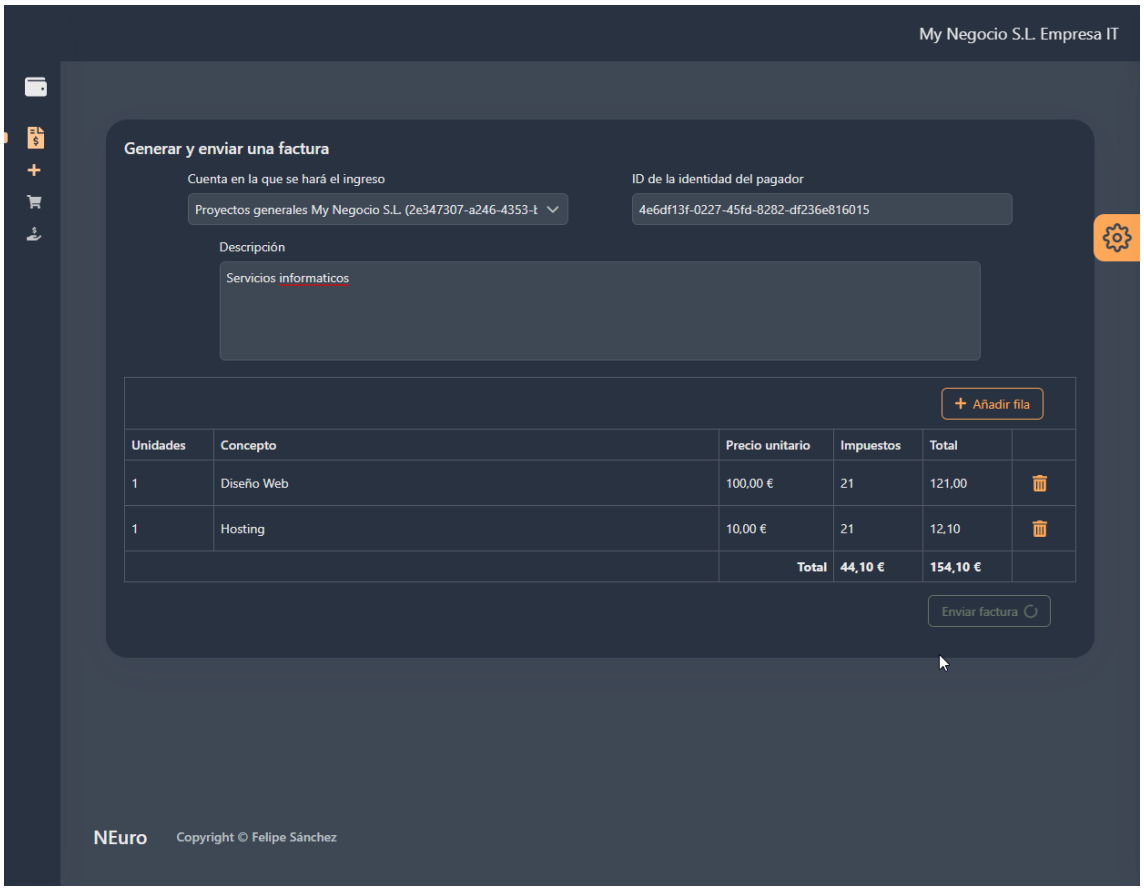
3.6.1 EMITIR FACTURA

ROLES: Empresa

Hay que dirigirse a: Facturas -> Emitir factura.



Una vez que rellenamos todos los datos de la factura podemos enviarla:

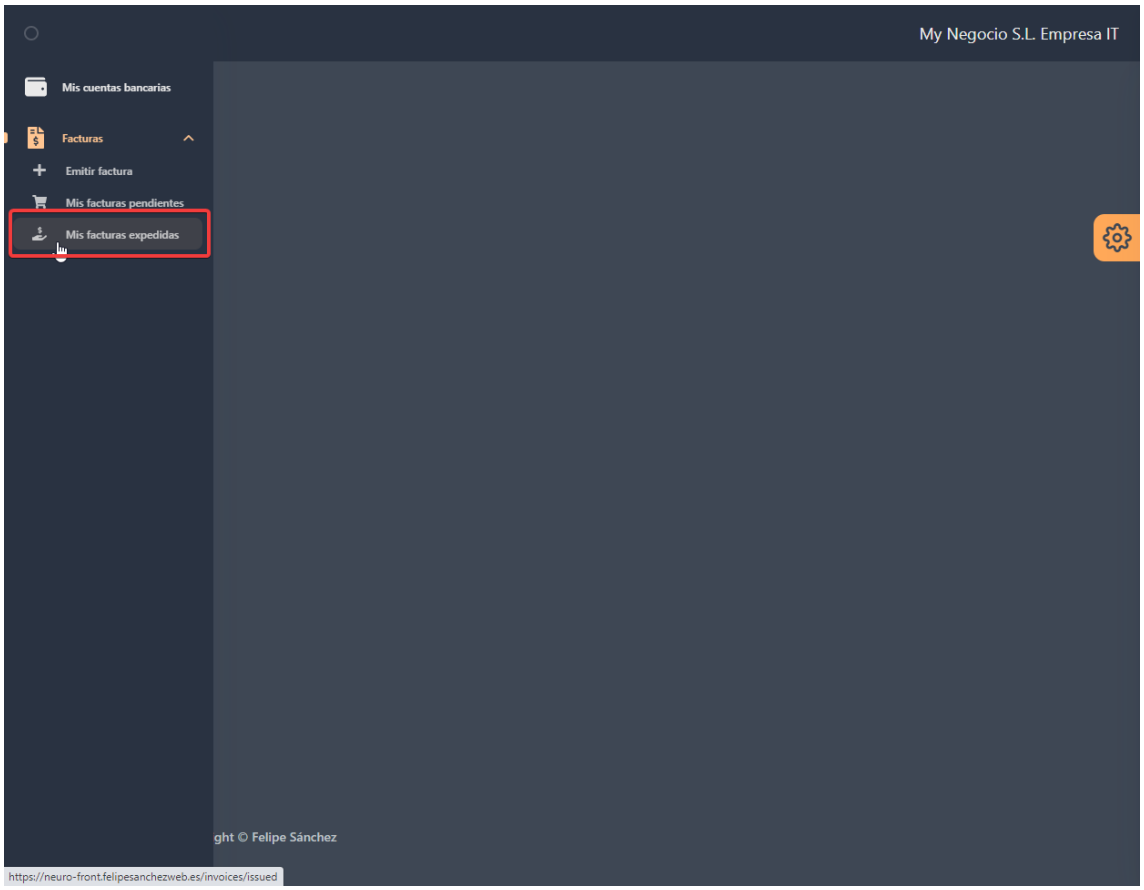


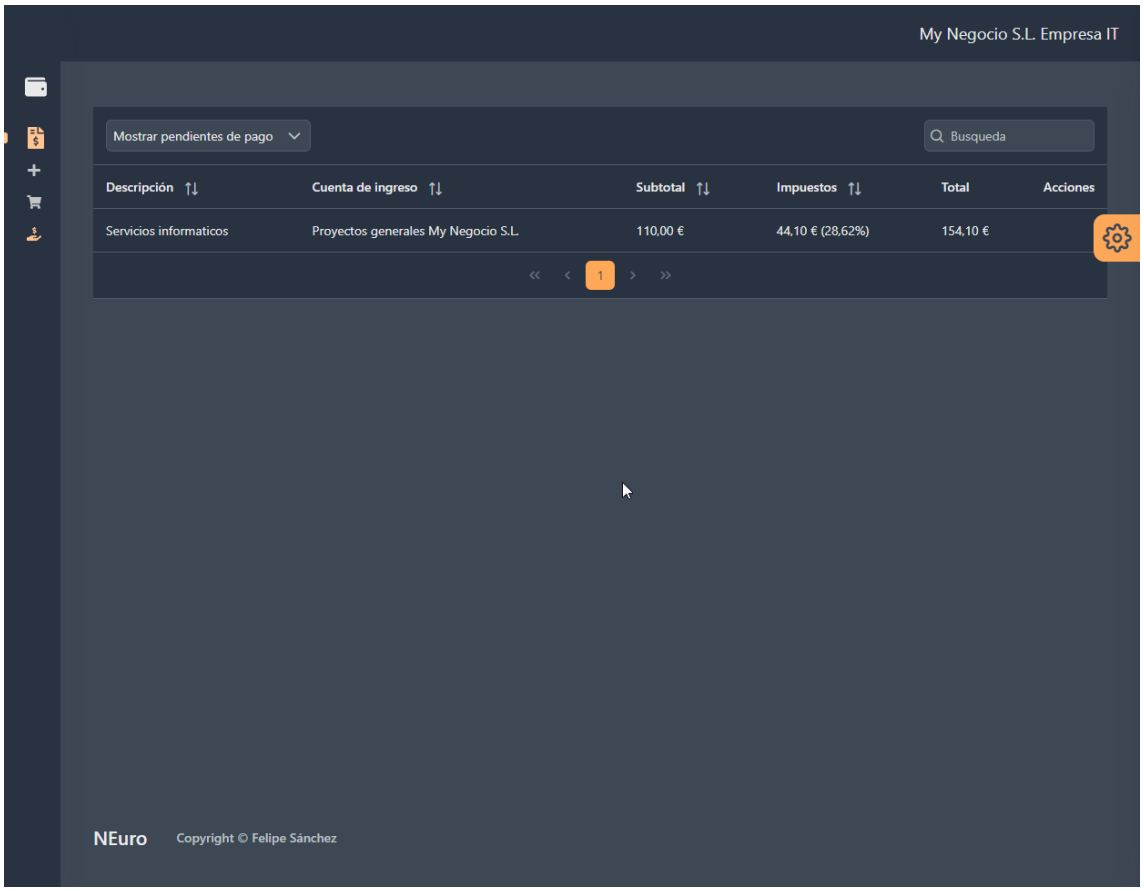
La factura se expedirá y los campos del formulario se limpiarán completamente

3.6.2 VER FACTURAS EMITIDAS

ROLES: Empresa

Para ver las facturas emitidas y su estado basta con dirigirse a: Facturas -> Mis facturas expedidas.



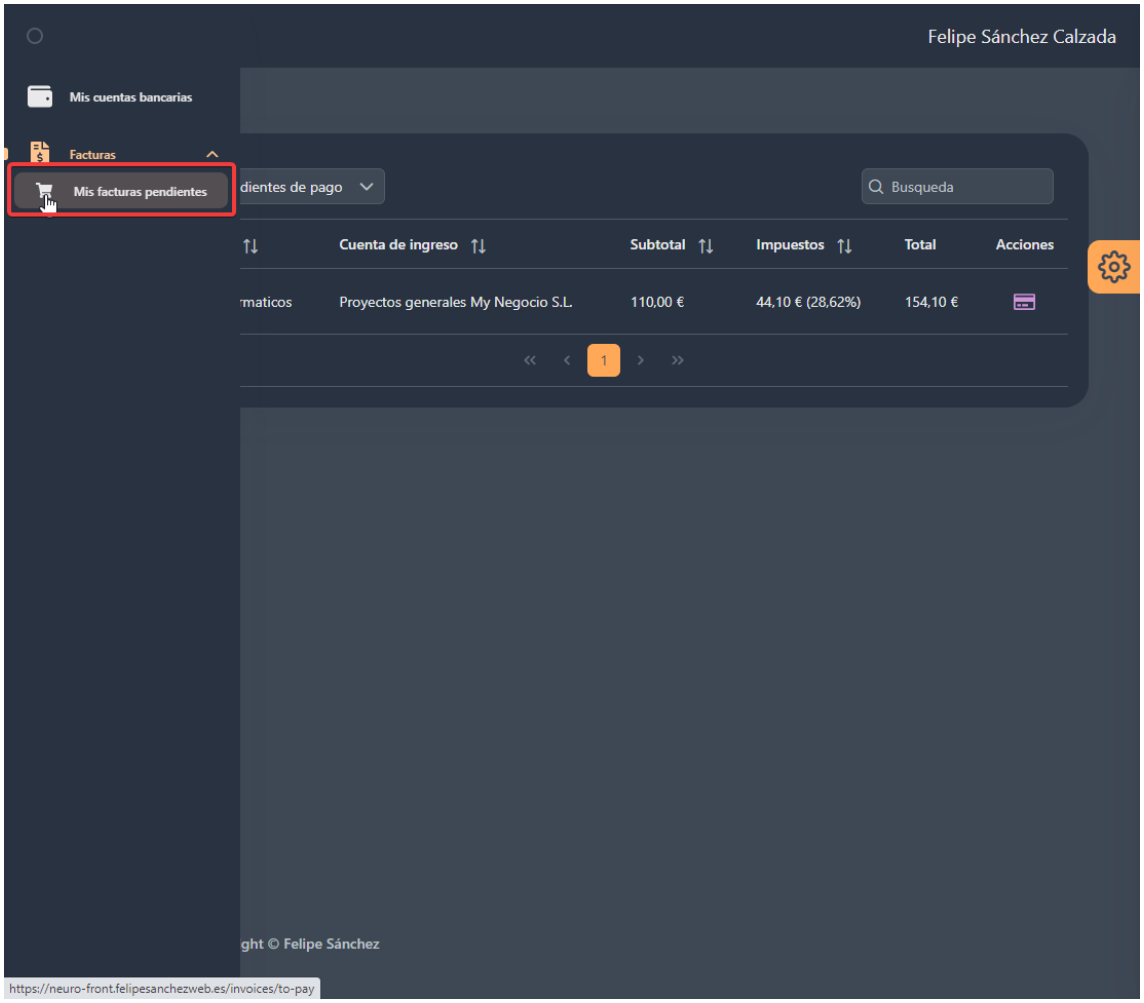


Podemos filtrar las facturas según su estado

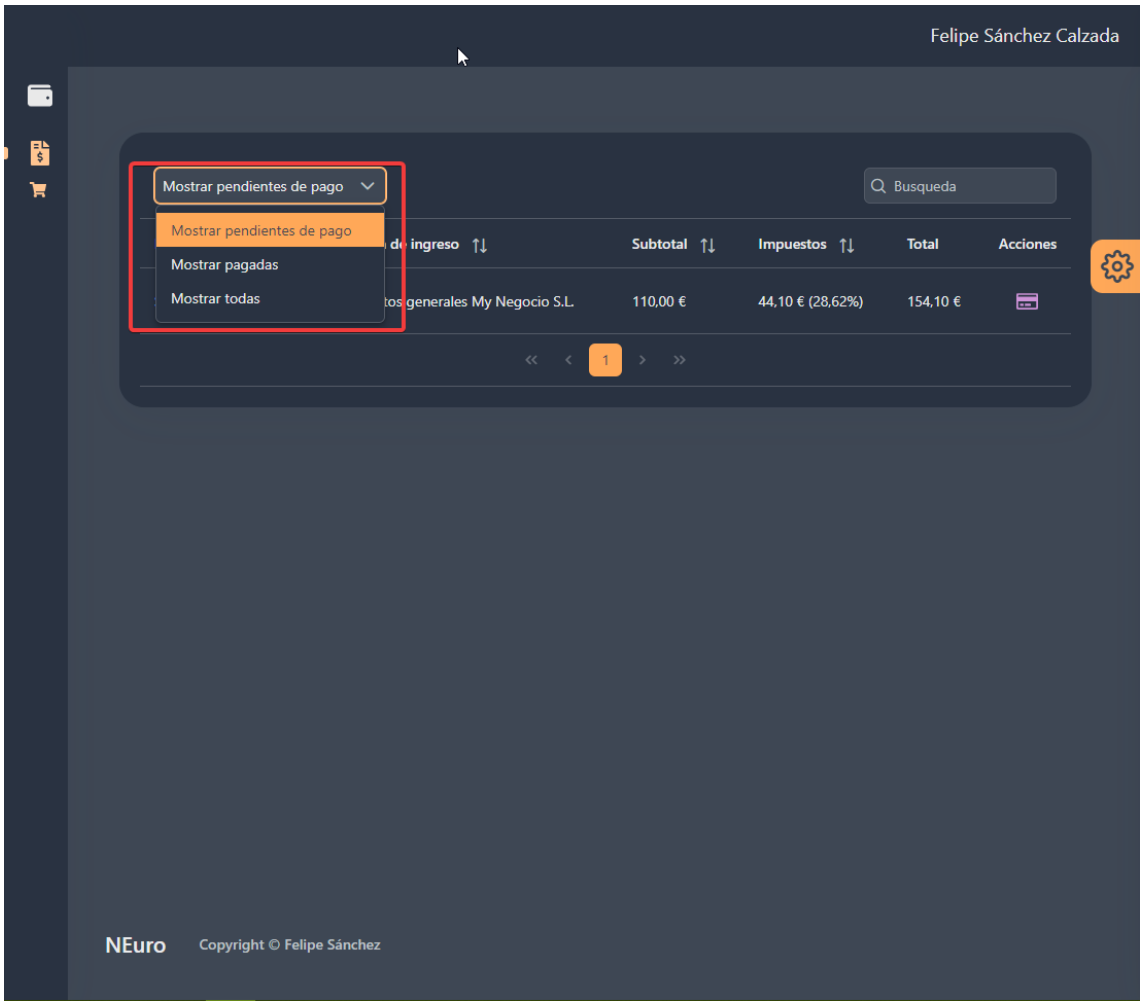
3.6.3 VER MIS FACTURAS PENDIENTES DE PAGO

ROLES: Usuario, Empresa, Hacienda

Hay que dirigirse a : Facturas -> Mis facturas pendientes



Se mostrará una tabla con las facturas y se podrá filtrar por su estado.

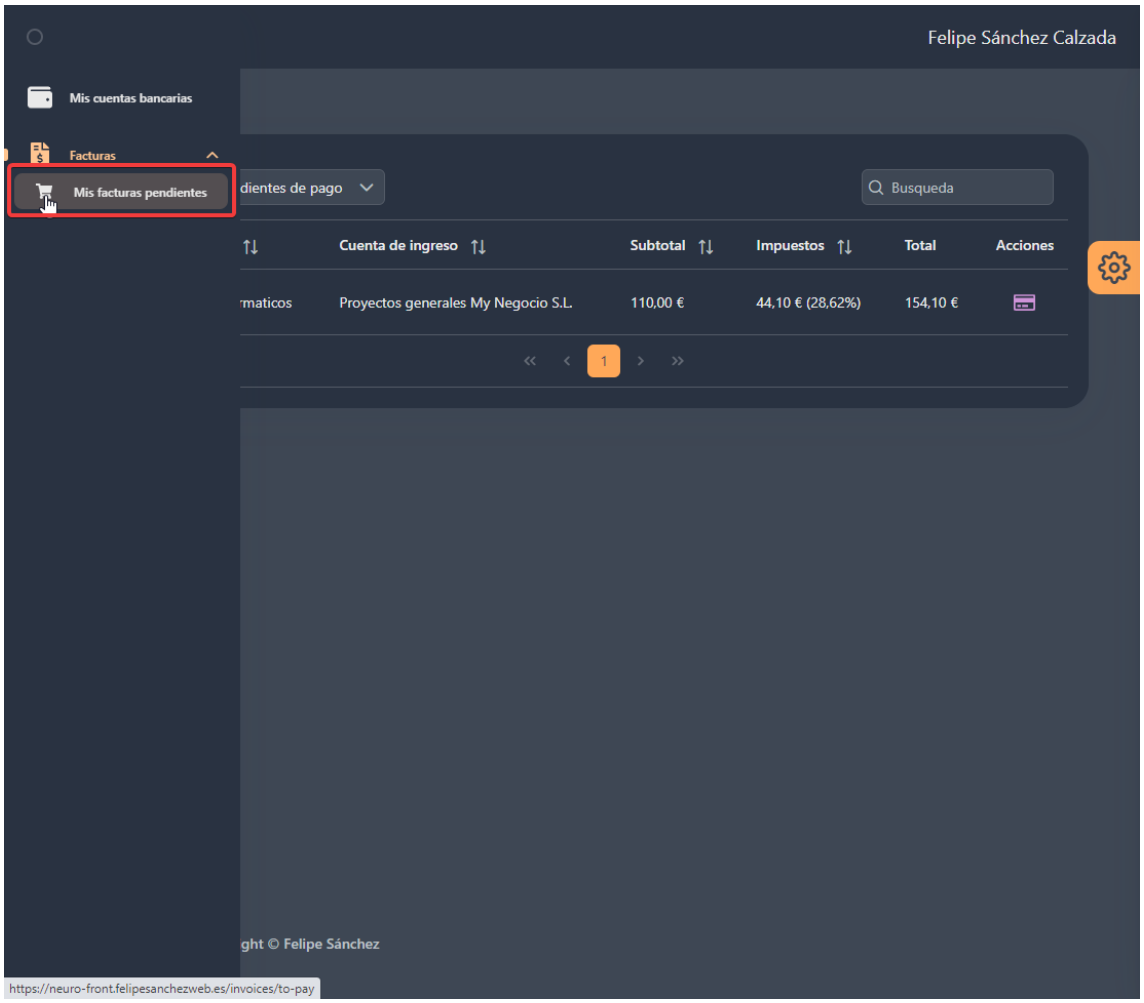


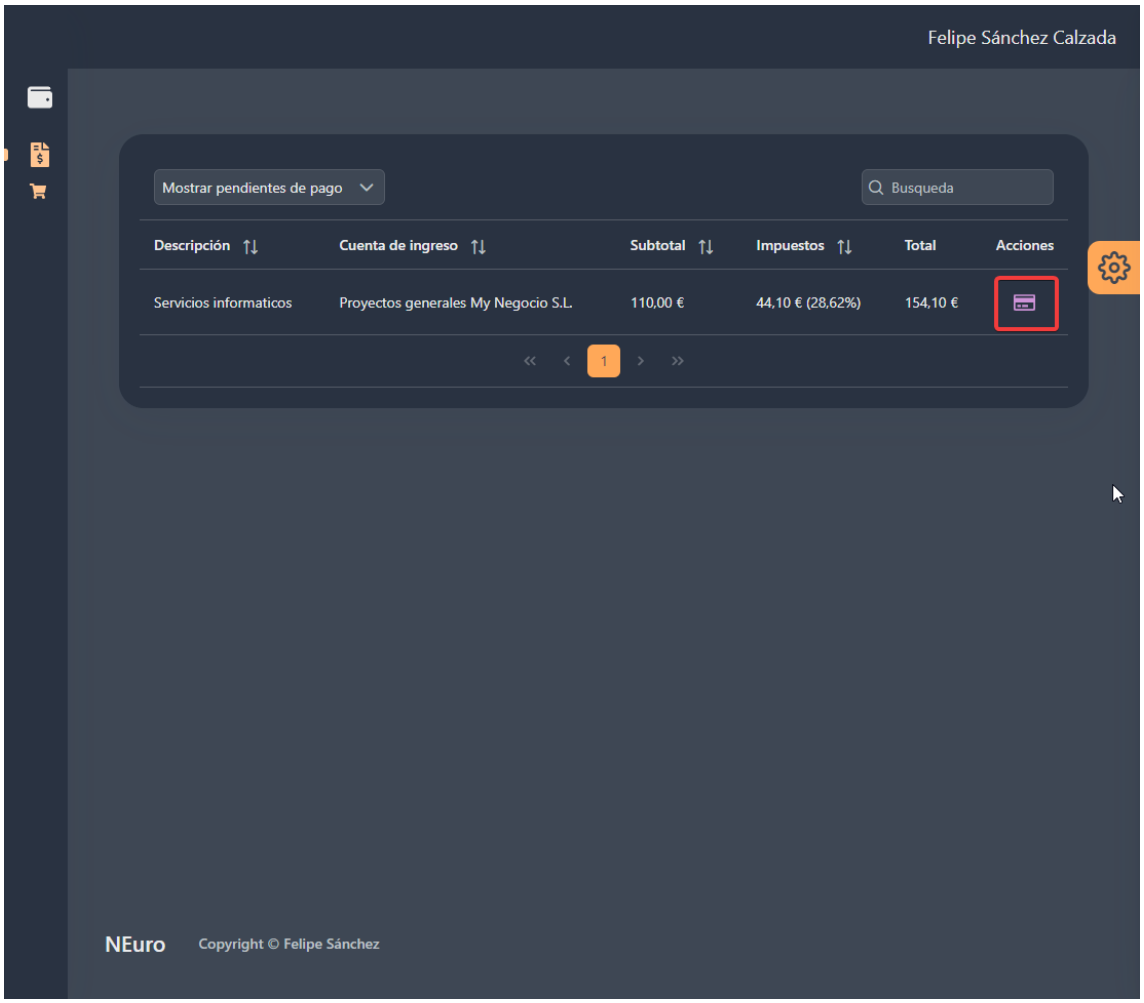
3.6.4 PAGAR UNA FACTURA PENDIENTE

ROLES: Usuario, Empresa, Hacienda Publica

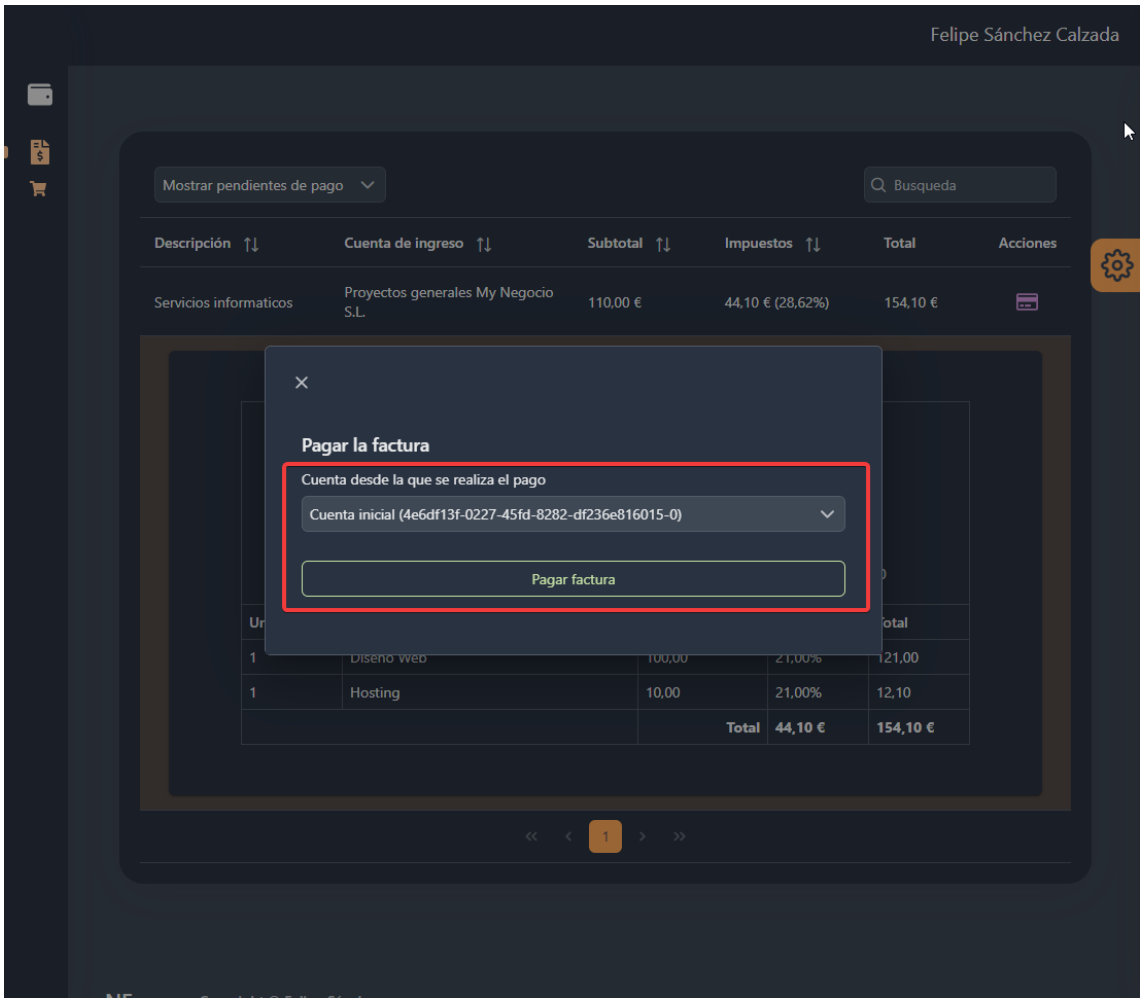
Para pagar una factuara hay que acceder a:

Facturas -> Mis facturas pendientes -> Click en el icono -> Seleccionar la cuenta desde la que se va a hacer el pago


Felipe Sánchez Calzada
Mis cuentas bancarias
Facturas
Mis facturas pendientes
dientes de pago
Busqueda
Cuenta de ingreso
Subtotal
Impuestos
Total
Acciones
rmaticos
Proyectos generales My Negocio S.L.
110,00 €
44,10 € (28,62%)
154,10 €
1
ght © Felipe Sánchez
https://neuro-front.felipesanchezweb.es/invoices/to-pay



Posterioprmente se selecciona la cuenta desde la que se va a realizar el pago y se hace click en pagar factura



4 ACCIONES MANUALES

Las acciones manuales son algunas operaciones que no están disponibles desde la web, y que por tanto es necesario hacer las llamadas a la API REST manuelmente.

4.1 CREACIÓN DE HACIENDA

URL: /tax-authorities

Método: POST

Cuerpo:

```
{
    "username": "hacienda-española",
    "email": "hacienda-española@c.c",
    "name": "Hacienda Pública España",
    "surname": " "
}
```

Authenticacion: Token JWT Bearer

4.2 CREACIÓN DE ADMINISTRADOR DE IDENTIDADES

URL: /identity-admins

Método: POST

Cuerpo:

```
{
    "username": "admin-username",
    "email": “admin-username @e.com",
    "name": "Admin",
    "surname": "Publico"
}
```

Authenticacion: Token JWT Bearer