COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA DISTRIBUIDO DE GESTIÓN SOBERANA PARA EL PAGO AUTOMÁTICO DE IMPUESTOS

Autor: Guillermo Fernández Pérez

Director: Óscar Sanz Sebastián

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Sistema Distribuido de Gestión Soberana para el pago automático de impuestos
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico 2022/2 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos.
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido
tomada de otros documentos está debidamente referenciada.

Fdo.: Guillermo Fernández Pérez Fecha: 28/ 08/ 2023

Autorizada la entrega del proyecto
EL DIRECTOR DEL PROYECTO

SANZ SEBASTIAN OSCAR - 70246746P

Firmado digitalmente por SANZ SEBASTIAN OSCAR - 70246746P
Fecha: 2023.08.29 08:04:25 +02'00'

Fdo.: Óscar Sanz Sebastián Fecha: 28/ 08/ 2023

Sistema Distribuido de Gestión Soberana para el Pago Automático de Impuestos

Autor: Fernández Pérez, Guillermo.

Director: Sanz Sebastián, Óscar.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen del Proyecto

El proyecto está basado en el concepto de gestión soberana (potestad de que el usuario sea el responsable de sus propias credenciales y permisos). Consiste en **la implementación de una plataforma única desde la cual los ciudadanos puedan delegar o revocar permisos a las distintas administraciones públicas, para que estas puedan cobrarles automáticamente sus recibos, multas o impuestos** en la cuenta bancaria que el usuario seleccione.

Palabras clave: Gestión soberana, Blockchain, autorización, Ethereum, Administración, Python, SQL, API.

1. Introducción

En la era digital actual, la relación con la información es un tema fundamental y se plantean continuamente cuestiones sobre privacidad y control. Este proyecto se centra en la intersección de tecnología y autodeterminación, proponiendo soluciones para otorgar el control a los ciudadanos en la gestión de su identidad digital. La gestión soberana, que trasciende lo tecnológico, aborda libertad y autonomía. La motivación radica en simplificar el pago de impuestos y multas, mejorando la experiencia del usuario. La centralización agiliza trámites complejos. Sin embargo, surge la preocupación por la seguridad y privacidad de los datos. La gestión soberana surge como respuesta, permitiendo a los usuarios control total y transparencia. El proyecto busca empoderar a los ciudadanos en el mundo digital, manteniendo la seguridad y adaptándose a cambios reguladores.

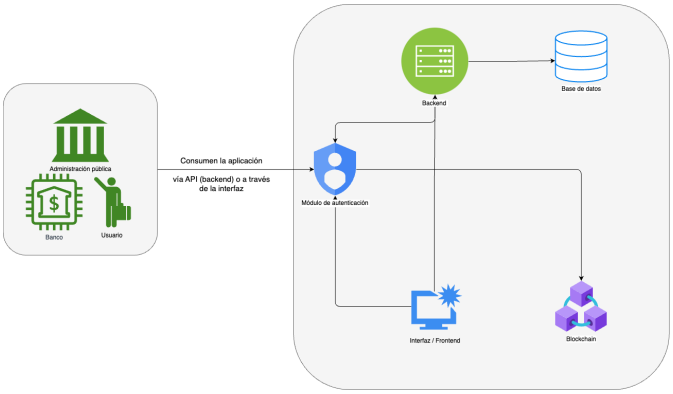
2. Definición del proyecto

En un mundo digitalizado, algunos sectores, como la administración pública, luchan por adaptarse. La complejidad y los procesos manuales persisten a pesar de la digitalización. Este TFG propone usar la tecnología para restaurar la confianza y la transparencia del ciudadano. Ofrece automatización y autorización ciudadana para equilibrar eficiencia y autonomía. La autenticación web3.0 asegura la privacidad a través de Ethereum. El proyecto busca eficiencia sin comprometer transparencia, integridad y privacidad.
Los principales objetivos que se definen son mejorar la experiencia del usuario, aumentar la seguridad y el control de acceso, facilitar la interoperabilidad, mejorar la eficiencia y escalabilidad, promover la adopción de estándares y gestionar la delegación de permisos.

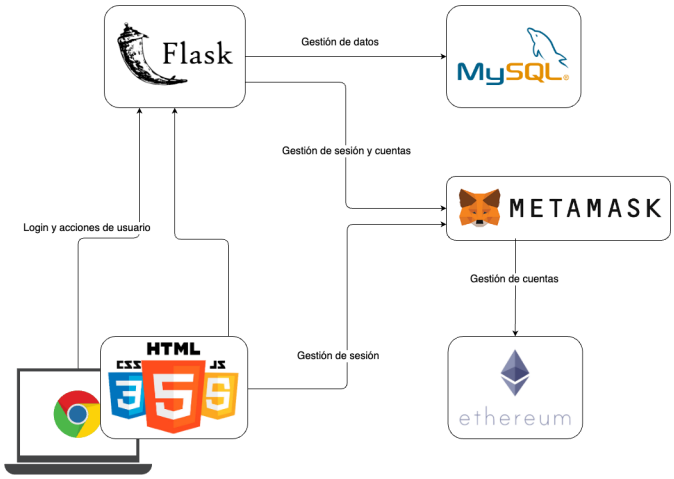
3. Descripción del sistema

Las tecnologías empleadas por cada uno de los componentes de la aplicación son las siguientes:

Módulo de autenticación	Metamask
Backend	Flask (Python)
Frontend	HTML, CSS y Javascript
Blockchain	Ethereum
Base de datos	SQLite y MySQL



Arquitectura global



Arquitectura de software

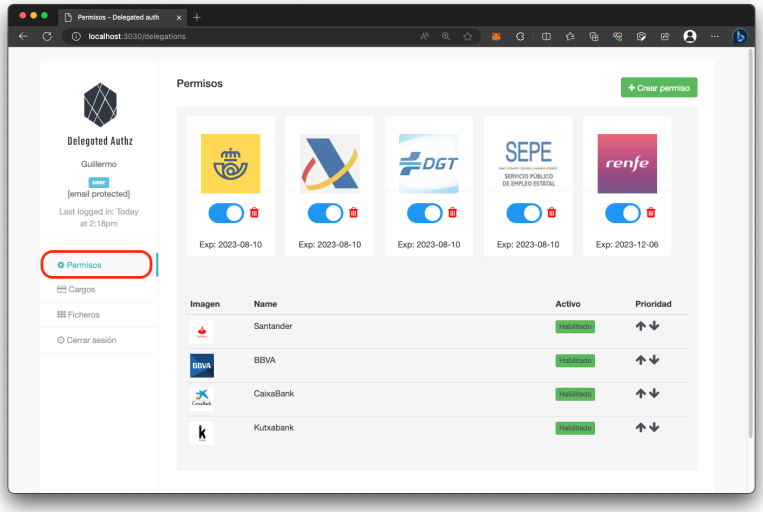
4. Resultados

A continuación, se detalla el análisis de resultados del proyecto en base al planteamiento inicial que se hizo sobre los objetivos generales del mismo.

El primer objetivo era crear un sistema automático, transparente y fiable. Aquí, la implementación de la tecnología Blockchain resultó en una mayor transparencia y fiabilidad en las transacciones. La automatización se logró al centralizar los procesos de pago, y se planean mejoras adicionales, como la migración completa a la plataforma Ethereum.

El otro de los objetivos generales se centraba en respetar la privacidad y la propiedad de la información del usuario. Mediante el uso de la blockchain y la autenticación web3.0, se estableció un sistema sólido que respeta

la privacidad del usuario. El grado de cumplimiento de este objetivo tiene margen para ser todavía mayor, apuntando hacia una integración completa con Ethereum en el futuro.



Vista de la sección 'Permisos'

5. Conclusiones

El proyecto realizado involucró la integración de herramientas innovadoras con sistemas convencionales. La adopción de una red de blockchain local fue clave, brindando seguridad y base para escalabilidad futura. Destacó la introducción de un nuevo método de inicio de sesión, incrementando la seguridad y explorando nuevas formas de verificación. Equilibrar la experiencia del usuario con un sistema robusto fue fundamental, combinando facilidad de uso y seguridad. La adaptabilidad se enfatizó al usar herramientas modernas para gestionar información. La investigación abarcó tecnología, preocupaciones sociales y problemas en la administración pública. El proyecto logró sus objetivos de crear un sistema de autorización de cobros, basado en web3 y Ethereum.

Sin ser todavía comercializable, ya representa una solución sólida para la gestión de datos y permisos en línea, devolviendo el control al usuario. Se espera que continúe y se desarrollen las mejoras propuestas en la presente memoria, consolidando la idea inicial.

6. Referencias (Bibliografía ampliada en el Capítulo 11 del presente documento)

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
- Zohar, A. (2015). *Bitcoin: under the hood*. Communications of the ACM, 58(9), 104–113.
- Allen, C. (2016). *The Path to Self-Sovereign Identity*. Coindesk.
- Hardman, D. (2018). *Sovrin's Token: A Stake in the Ground*. Sovrin Foundation White Paper.

Distributed Sovereign Management System for Automatic Tax Payment

Author: Fernández Pérez, Guillermo.

Supervisor: Sanz Sebastián, Óscar.

Collaborating Entity: ICAI – Universidad Pontificia Comillas.

Abstract

The project is based on the concept of sovereign management (the ability of the user to be responsible for their own credentials and permissions). It consists of **the implementation of a single platform from which citizens can delegate or revoke permits to the different public administrations**, so that they can automatically collect their receipts, fines or taxes in the bank account that the user selects.

Keywords: Sovereign management, Blockchain, authorization, Ethereum, Administration, Python, SQL, API.

1. Introduction

In today's digital age, the relationship with information is a fundamental issue and questions about privacy and control are constantly raised. This project focuses on the intersection of technology and self-determination, proposing solutions to give citizens control in the management of their digital identity. Sovereign management, which transcends technology, addresses freedom and autonomy. The motivation lies in simplifying the payment of taxes and fines, improving the user experience. Centralization streamlines complex procedures. However, concerns arise for data security and privacy. Sovereign management emerges as a response, allowing users full control and transparency. The project seeks to empower citizens in the digital world, maintaining security and adapting to regulatory changes.

2. Project definition

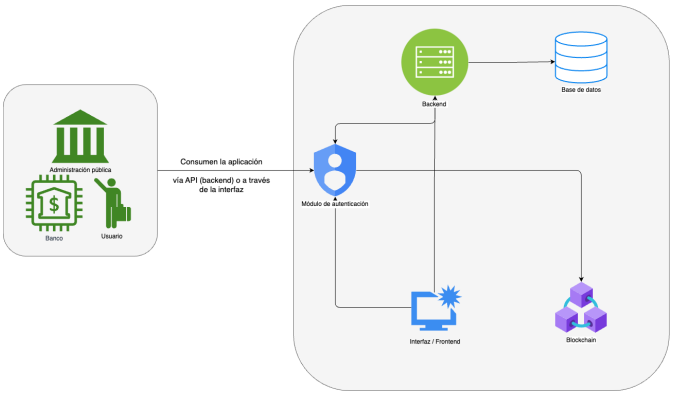
In a digitalized world, some sectors, such as public administration, struggle to adapt. Complexity and manual processes persist despite digitalization. This TFG proposes to use technology to restore citizen confidence and transparency. It offers automation and citizen authorization to balance efficiency and autonomy. Web3.0 authentication ensures privacy through Ethereum. The project seeks efficiency without compromising transparency, integrity and privacy.

The main objectives that are defined are to improve the user experience, increase security and access control, facilitate interoperability, improve efficiency and scalability, promote the adoption of standards and manage the delegation of permissions.

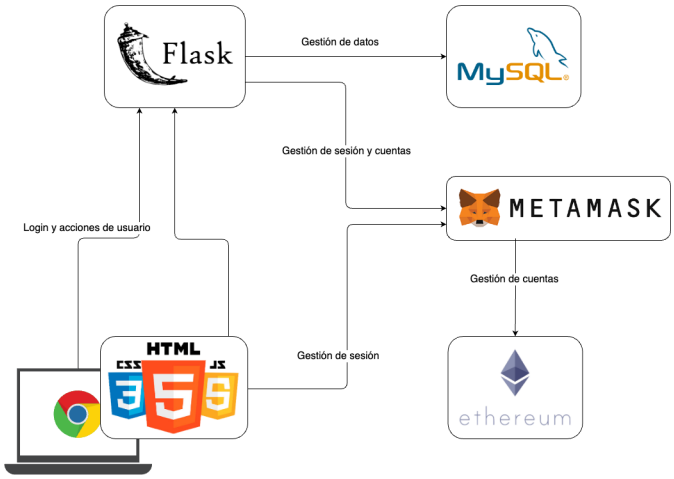
3. System description

The technologies used by each of the components of the application are the following:

Authentication module	Metamask
Backend	Flask (Python)
Frontend	HTML, CSS and Javascript
Blockchain	Ethereum
Database	SQLite and MySQL



Global architecture



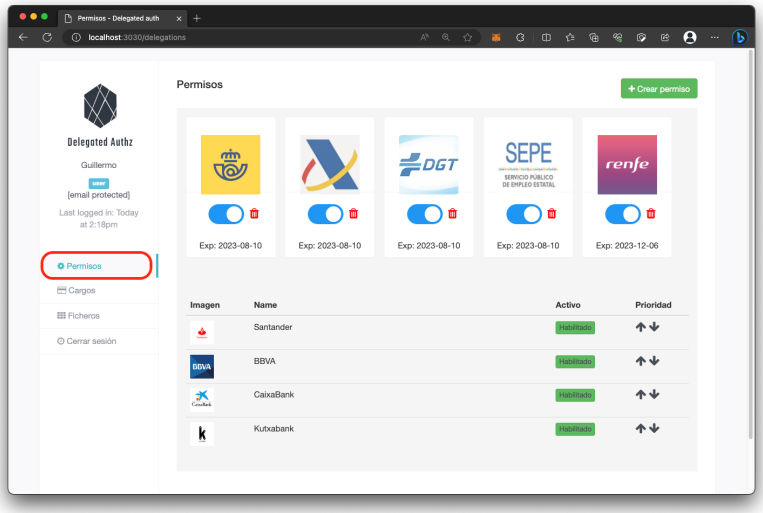
Software architecture

4. Results

Next, the analysis of the results of the project is detailed based on the initial approach that was made on the general objectives of the same.

The first objective was to create an automatic, transparent and reliable system. Here, the implementation of Blockchain technology resulted in greater transparency and reliability in transactions. Automation was achieved by centralizing payment processes, and further improvements are planned, such as full migration to the Ethereum platform.

The other general objective focused on respecting the privacy and ownership of user information. Through the use of blockchain and web3.0 authentication, a solid system was established that respects user privacy. The degree of fulfillment of this objective has room to be even greater, pointing towards a complete integration with Ethereum in the future.



View of the 'Permissions' section

5. Conclusions

The project involved the integration of innovative tools with conventional systems. The adoption of a local blockchain network was key, providing security and foundation for future scalability. He highlighted the introduction of a new login method, increasing security and exploring new forms of verification. Balancing the user experience with a robust system was critical, combining ease of use and security. Adaptability was emphasized when using modern tools to manage information. The research covered technology, social concerns and problems in public administration. The project achieved its goals of creating a collection authorization system, based on web3 and Ethereum.

Without yet being marketable, it already represents a solid solution for online data and permissions management, returning control to the user. It is expected that the improvements proposed in this report will continue and be developed, consolidating the initial idea.

6. References (Bibliography in Chapter 11 of this document)

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
- Zohar, A. (2015). *Bitcoin: under the hood*. Communications of the ACM, 58(9), 104–113.
- Allen, C. (2016). *The Path to Self-Sovereign Identity*. Coindesk.
- Hardman, D. (2018). *Sovrin's Token: A Stake in the Ground*. Sovrin Foundation White Paper.

Índice de la memoria

Índice de figuras

1 Introducción

La revolución digital ha transformado fundamentalmente nuestra relación con la información. A medida que navegamos por esta era de interconexión sin precedentes, nos encontramos en una encrucijada: mientras que las posibilidades tecnológicas se expanden a una velocidad vertiginosa, las preocupaciones sobre la privacidad, la seguridad y la autonomía de nuestros datos personales siguen siendo persistentes. En este paisaje cambiante, surge una pregunta esencial: ¿quién tiene realmente el control y la propiedad de nuestra identidad digital?

Este proyecto se enmarca en esta problemática vital, buscando ofrecer soluciones en la intersección de tecnología y autodeterminación. Al sumergirnos en conceptos como la "gestión soberana", nos proponemos trascender los modelos tradicionales y ofrecer a los individuos un control más tangible sobre su propia información, un derecho que se ha tornado cada vez más esencial en nuestro mundo contemporáneo.

La gestión soberana, en su esencia, representa un cambio de paradigma. Va más allá de simples cuestiones tecnológicas y aborda preocupaciones fundamentales sobre la libertad, la autonomía y el derecho inherente de cada individuo a determinar cómo se utiliza y comparte su identidad digital. Sin embargo, lograr esta visión requiere no solo una reflexión profunda sobre los ideales que buscamos, sino también una comprensión sólida de las herramientas y tecnologías que pueden hacerla realidad.
Esta introducción al estado del arte del proyecto proporciona un panorama del ecosistema actual, estableciendo las bases para una exploración en profundidad de las soluciones emergentes, las tecnologías revolucionarias y las visiones innovadoras que están redefiniendo cómo concebimos y gestionamos nuestra presencia digital en el siglo XXI.

1.1 Motivación del proyecto

La motivación detrás del proyecto de gestión soberana en el pago de impuestos y multas es abordar los desafíos y dificultades que enfrentan los ciudadanos al interactuar con múltiples entidades públicas para cumplir con sus obligaciones fiscales y solventar sus deudas. El propósito fundamental es brindar una solución innovadora y eficiente que simplifique y agilice este proceso, permitiendo a los usuarios realizar pagos de manera rápida y segura.

Una de las principales motivaciones es mejorar la experiencia del usuario al enfrentarse a los trámites relacionados con los impuestos y multas. Actualmente, los ciudadanos deben lidiar con una serie de complicaciones, como la necesidad de visitar diferentes oficinas gubernamentales, presentar numerosos documentos y realizar múltiples transacciones bancarias para cumplir con sus obligaciones. Esto puede resultar tedioso, consumir tiempo y generar confusión en el proceso. El proyecto busca cambiar esta realidad al proporcionar una plataforma centralizada que permita a los usuarios gestionar y pagar sus impuestos y multas desde un solo lugar, simplificando así todo el procedimiento.

Esta automatización y facilidad, trae consigo otro problema mayor el de la seguridad y la privacidad de la información. En un mundo digital donde los datos personales se han convertido en una moneda de cambio, garantizar su seguridad y privacidad es primordial. Las filtraciones de datos, los ataques cibernéticos y el mal uso de la información son preocupaciones legítimas que los ciudadanos tienen cuando se trata de compartir sus detalles financieros y personales en línea.

Aquí es donde la gestión soberana cobra una relevancia crucial. Al permitir que los usuarios tengan un control total sobre sus propios datos, eliminando intermediarios innecesarios y proporcionando transparencia en todas las transacciones, se pretende construir un sistema que inspire confianza. No se trata solo de simplificar un proceso, sino de empoderar al ciudadano en el mundo digital, otorgándole verdadera propiedad y dominio sobre su información.

La motivación de este proyecto se basa en un deseo profundo de mejorar la interacción de los ciudadanos con las entidades fiscales, de otorgarles más control sobre sus datos y de adaptarse a un mundo digital en constante cambio, todo ello mientras se enfrenta a los desafíos inherentes de la seguridad, la privacidad y la evolución regulatoria.

1.2 Contexto histórico

El entorno fiscal y regulatorio está en constante evolución. Las leyes cambian, las obligaciones fiscales se adaptan y los procedimientos gubernamentales se actualizan. Para muchos ciudadanos, mantenerse al día con estas constantes modificaciones puede ser un desafío en sí mismo. Por lo tanto, otra motivación esencial detrás de este proyecto es crear una solución dinámica y adaptable que pueda evolucionar junto con el entorno regulatorio, garantizando que los usuarios siempre estén al tanto de sus responsabilidades y derechos.

En este contexto contemporáneo, los datos se han convertido en uno de los activos más valiosos. Cada click, cada compra online, cada "me gusta" en una red social se traduce en información que, una vez recopilada y analizada, tiene un valor incalculable. Las organizaciones, grandes y pequeñas, buscan constantemente aprovechar estos datos para obtener ventajas competitivas, crear experiencias personalizadas para los usuarios y, en muchos casos, obtener beneficios económicos. La información, que alguna vez estuvo contenida en voluminosos archivos y bibliotecas, ahora fluye en bytes y se almacena en la nube, creando una economía en sí misma donde los datos son la moneda principal.

Finalmente, vivimos en una era de innovación tecnológica. Los avances en el campo de la tecnología de blockchain, la web 3.0 y las soluciones de autorización delegada presentan oportunidades inexploradas para redefinir la forma en que interactuamos con las instituciones. Adoptar estas tecnologías emergentes no solo puede mejorar la eficiencia y la transparencia, sino que también puede posicionar al sistema como un referente a nivel global en la gestión fiscal digital.

1.3 Problemas actuales

Sin embargo, este inmenso progreso no ha venido exento de desafíos. Uno de los más palpables ha sido la pérdida de privacidad. Las grandes corporaciones e instituciones, en su afán de recolectar datos, a menudo traspasan límites éticos y legales en cuanto a cómo obtienen y usan la información personal de los usuarios o ciudadanos.

Aunque muchos servicios online se ofrecen de manera "gratuita", en realidad, el precio que se paga es el acceso a datos personales que luego son utilizados para fines publicitarios, comerciales y, en ocasiones, vendidos a terceros sin el conocimiento explícito del usuario. En el contexto de las instituciones, el fin no es monetizar es información, pero si afecta de una forma más peligrosa aún sobre las libertades individuales de un sujeto y sus derechos como ciudadano.

Esta dinámica ha llevado a que la privacidad se vea amenazada y que muchos individuos sientan que están bajo constante vigilancia en la red.

A esto se suma la falta de control y autonomía sobre la información personal. El usuario promedio a menudo se encuentra en una situación en la que, aunque genera y proporciona datos, no tiene un control real sobre ellos. Una vez que se comparten en una plataforma o servicio, estos datos quedan fuera del alcance del individuo, y las decisiones sobre cómo se almacenan, se comparten o se utilizan quedan en manos de las corporaciones o instituciones. Esta dinámica ha llevado a una creciente sensación de impotencia y a la percepción de que, en el vasto mundo digital, el individuo ha perdido su soberanía sobre su propia información.

2 Estado de la Cuestión

Durante el siglo XXI, la humanidad ha experimentado un salto evolutivo en su relación con la tecnología. Los datos, ese intangible recurso, han emergido como uno de los activos más valiosos de nuestra era, transformando no solo nuestra economía, sino también nuestra percepción del valor, propiedad y privacidad. Este cambio tan abrupto ha provocado la necesidad de reconsiderar y remodelar cómo interactuamos, controlamos y, sobre todo, cómo confiamos en las entidades digitales y las infraestructuras que las respaldan. En este contexto, emerge este proyecto, anclado en la confluencia de tecnologías revolucionarias y conceptos innovadores que buscan redefinir la administración, autorización y propiedad de la información.

La gestión soberana, un término que se ha solidificado en la conciencia colectiva en los últimos años, alude a la autonomía sobre nuestra identidad digital y a cómo esta puede ser gestionada de manera descentralizada. Este concepto, aunque profundamente arraigado en ideales de libertad y autodeterminación, no ha surgido en el vacío. En su lugar, es producto de avances tecnológicos y una respuesta a las crecientes preocupaciones sobre la privacidad, seguridad y monopolio de datos.

El estado del arte, por lo tanto, no solo se refiere a lo más avanzado o contemporáneo en un campo particular, sino también a la comprensión y contextualización de cómo hemos llegado hasta aquí y hacia dónde podríamos dirigirnos. Es un testimonio de la innovación humana y, en el caso de este proyecto, refleja una convergencia de visiones, tecnologías y soluciones que buscan un mundo digital más seguro, transparente y equitativo.

La exploración en el estado del arte aborda, en detalle, cada concepto utilizado en el proyecto, desde la gestión soberana, pasando por la Web 3.0 y la solidez inmutable del blockchain, hasta la autorización delegada. Esta ruta es esencial no solo para entender el proyecto, sino también para trazar un camino hacia un futuro en el que cada individuo tiene un control auténtico y sin precedentes sobre su propia ser digital.

2.1 Gestión Soberana

La gestión soberana no es un concepto que haya emergido de la noche a la mañana. La raíz de este paradigma radica en la creciente conciencia de la necesidad de la propiedad individual sobre la información en un mundo cada vez más interconectado.

Desde la aparición de los primeros sistemas de gestión de bases de datos en los años 60 y 70, las personas empezaron a entender el potencial (y el peligro) de tener su información almacenada en lugares que no controlaban. A medida que avanzaba la revolución digital, este entendimiento se volvió más urgente y palpable.

Algunos de los principales desarrollos y movimientos significativos en el campo son:

- **Derechos Digitales**: A finales del siglo XX y principios del XXI, comenzaron a surgir movimientos que abogaban por derechos digitales. Estos movimientos luchaban contra la vigilancia digital, la recopilación no consensuada de datos y abogaban por una mayor privacidad en línea.

- **Legislaciones de Protección de Datos**: La aprobación de legislaciones como el GDPR en Europa marcó un hito en el reconocimiento del control de datos personales como un derecho humano fundamental.

- **Emergencia de Comunidades**: Surgieron comunidades y organizaciones que defendían la idea de una gestión de datos descentralizada, alejándose del control de corporaciones y gobiernos. Estas comunidades, en gran parte de los casos, apoyan proyectos de software libre que potencian y aceleran la evolución de la tecnología hacia un futuro más deseable.

2.1.1 Beneficios de la Gestión Soberana

Sin lugar a duda, los beneficios son difíciles de discutir. Este cambio, tanto ideológico como técnico aportaría los siguientes beneficios:

- **Autonomía Personal**: Los individuos no solo tienen control sobre sus datos, sino que también ganan una sensación de autonomía y empoderamiento en el espacio digital.

- **Seguridad Mejorada**: Al reducir los puntos centralizados de falla y al asegurarse de que los datos se compartan solo cuando sea necesario, se reduce el riesgo de brechas caída del servicio y la seguridad del dato.

- **Confianza Reforzada**: Las transacciones e interacciones en línea ganan en confianza cuando los usuarios saben que sus datos están protegidos y no se utilizan sin su consentimiento. Además de que este sistema potencia enormemente la transparencia, pilar fundamental en cualquier tipo de relación.

2.1.2 Desafíos de la Gestión Soberana:

Como casi en todo, la teoría es fácil de defender, pero la aplicación práctica suele ser más compleja. En este caso, se identifican los principales problemas y retos a los que se enfrentan este tipo de transformaciones:

- **Educación y Comprensión**: No todos tienen el conocimiento técnico para gestionar su información de manera efectiva. Es esencial que haya herramientas y recursos educativos adecuados. Hoy en día los conceptos sobre los que se apoyan este tipo de proyectos son desconocidos por el público general, que, a su vez, ya vive en un entorno de brecha digital, la

cual, empieza a excluir a la gente mayor de este tipo de aplicaciones. Sin duda, uno de los mayores retos, por eso este tipo de soluciones tendría que operar de la forma más sencilla y clara posible.

- **Adopción Masiva**: La gestión soberana es contraria a muchos modelos de negocio actuales que dependen de la recopilación y monetización de datos, teniendo que abrirse paso en un entorno en el que las grandes corporaciones controlan el mercado y el futuro de la tecnología. La concienciación del usuario, como mayor agente del cambio, sería fundamental, si no cambia la demanda del mercado hacia soluciones más respetuosas con la privacidad las empresas no van a cambiar de modelo de negocio (o por lo menos no a la velocidad deseada).

- **Interoperabilidad**: La gestión de datos de forma soberana debe ser interoperable para que los individuos puedan interactuar libremente en diferentes plataformas y servicios sin perder el control sobre sus datos. Es decir, tu como usuario posees tu información y en función de la aplicación o servicio con el que operes aceptas compartirla o no. Esto requiere de un cambio tecnológico importante en el que las aplicaciones alteren significativamente sus diseños para operar bajo este modelo.

2.1.3 Gestión Soberana en la Sociedad Actual

Más allá de la tecnología, la gestión soberana es un reflejo de un cambio cultural y social. Las generaciones más jóvenes, en particular, están más conscientes de su presencia digital y de los riesgos asociados con la divulgación de información. Además, escándalos recientes relacionados con el mal uso de datos personales han impulsado el deseo de una mayor autonomía digital.

Un ejemplo común que está sirviendo para concienciar al usuario es la propiedad de las fotos, vídeos u otra información que compartimos en redes sociales, la cual, en algunas circunstancias pertenece a la empresa detrás de la aplicación. Además, toda esta información se utiliza para perfilar a el usuario para ofrecer información tan a medida que a veces se cree que los dispositivos nos escuchan, generando una sensación de control y vigilancia poderosa.

En el modelo actual, nosotros “navegamos” e interactuamos a través de distintos servicios, iniciamos sesión, chateamos, compartimos datos… etc., a través de plataformas las cuales almacenan todos estos datos. Aunque algunas normativas protegen la propiedad intelectual o los datos personales de los usuarios, nunca las posee el usuario (ni física ni digital).

Una gestión soberana en nuestra era digital implicaría un sistema en el que, como usuario, tuvieras almacenada toda tu información y decidieras como y con qué servicios compartirlas. Esto además de la propiedad actual de la información, también permite al usuario, en cualquier momento, revocar el permiso de acceso a esos datos.

Actualmente, uno de los mayores problemas en el entorno digital es el derecho al olvido, o la dificultad de borrar información de forma permanente en los servicios que consumimos a diario. Siendo propietarios, no solo jurídicamente, sino también teniendo la posesión digital de la información, se podrían garantizar una soberanía real de la información.

2.1.4 Implicaciones para el Proyecto

Al considerar la gestión soberana en el contexto de un sistema de pagos y gestión de permisos, se destaca la relevancia de garantizar que cada transacción y cada pieza de información compartida esté bajo el completo control del usuario. Esto no solo es esencial desde una perspectiva ética, sino que también puede ser un punto de venta único, diferenciando el proyecto de otras iniciativas similares estancadas en sistemas tradicionales.

La gestión soberana, más que una tendencia tecnológica, es una respuesta a una necesidad social emergente. En un mundo donde la información es poder, la capacidad de controlar y decidir sobre nuestros propios datos se vuelve esencial.

En este proyecto, el objetivo final es que tu como usuario seas dueño de tu identidad digital, los permisos a terceros y la capacidad de revocar cualquiera de los permisos que des a cualquier tercero en cualquier momento. Traducido al caso de uso del proyecto, el cual busca centralizar y automatizar el cobro de impuestos de la administración pública, significa que el acceso a tus datos bancarios; la creación, expiración y borrado de permisos; por último, y todas las configuraciones de la plataforma, no se alojan en el servidor de tus proveedores, sino en sistema descentralizado accesible únicamente por tu identidad digital.

La gestión soberana en un entorno digital necesita apoyarse en una base tecnológica y en unas metodologías específicas, las cuáles se alinean perfectamente con sus principios.

2.2 Web 3.0

La Web 3.0 es conocida comúnmente como la "web semántica". A diferencia de sus predecesoras, Web 1.0 y Web 2.0, que se centraban en la información y la interacción respectivamente, la Web 3.0 tiene como objetivo crear una internet más inteligente, personalizada y descentralizada, en un entorno en que las grandes corporaciones absorben la mayor parte del tráfico y propiedad de la información en internet.

Sus principales puntos fuertes y mejoras:

- **Descentralización:** En lugar de depender de entidades centralizadas para alojar información y servicios, la Web 3.0 se basa en redes distribuidas.

- **Interconexión:** Las aplicaciones y servicios en Web 3.0 están diseñados para operar de forma conjunta, lo que permite la interacción sin fisuras entre plataformas y sistemas.

- **Inteligencia Artificial:** La Web 3.0 integra capacidades de inteligencia artificial y machine learning para ofrecer experiencias más personalizadas a los usuarios.

- **Interoperabilidad:** Los datos se pueden compartir y utilizar entre diferentes aplicaciones y servicios gracias a estándares y tecnologías comunes.

- **Control de los datos:** Los usuarios tienen más control sobre sus propios datos, incluida la capacidad de determinar cómo y dónde se utilizan.

La Web 3.0 aún está en una fase de desarrollo y adopción, pero ha visto avances significativos en los últimos años, apoyada principalmente en:

- **Tecnologías de blockchain:** La descentralización es una característica clave de la Web 3.0 y blockchain es una de las principales tecnologías que la facilitan. Ethereum, por ejemplo, ha permitido la creación de aplicaciones descentralizadas (dApps) que operan en una red distribuida.

- **Búsquedas semánticas:** Las tecnologías de búsqueda están evolucionando para entender el contexto y la intención del usuario, en lugar de simplemente buscar palabras clave.

- **Plataformas de Identidad Digital:** Estas plataformas permiten a los usuarios tener una identidad digital única que puede ser utilizada en múltiples servicios, dándoles más control sobre sus datos personales.

- **Interacción avanzada:** Con el auge de la Realidad Virtual y Aumentada, la Web 3.0 está permitiendo formas más inmersivas de interactuar con contenido y servicios online.

2.2.1 Aplicación en el concepto de Gestión Soberana

La Web 3.0 es esencial para la gestión soberana por diversas razones:

- **Propiedad de los datos:** La descentralización inherente a la Web 3.0 permite a los usuarios tener un control y propiedad genuina sobre sus datos. Esto es fundamental para el concepto de gestión soberana, ya que permite a los individuos decidir quién tiene acceso a sus datos y con qué propósito.

- **Transacciones seguras:** Las tecnologías de blockchain que respaldan gran parte de la Web 3.0 ofrecen transacciones más seguras y transparentes. En el contexto de la gestión soberana, esto asegura que los pagos y otros intercambios se realicen de manera confiable.

- **Interoperabilidad:** Dado que la gestión soberana involucra la interacción con diversas entidades (gobiernos, instituciones financieras, etc.), la capacidad de la Web 3.0 para facilitar interacciones sin fisuras entre diferentes plataformas es crucial.

- **Contratos inteligentes:** Estos permiten la automatización de procesos y acuerdos basados en condiciones predefinidas. En la gestión soberana, podrían utilizarse para automatizar aspectos como el cobro de impuestos o la emisión de multas bajo ciertas condiciones.

En conjunto, la Web 3.0 proporciona las herramientas y capacidades necesarias para hacer realidad la visión de una gestión soberana, donde los individuos tienen control, seguridad y transparencia en sus interacciones con las entidades públicas y privadas.

2.3 Blockchain

Blockchain, también conocido como cadena de bloques, es una tecnología de registro distribuido donde los datos se almacenan en bloques y se enlazan en una cadena de forma secuencial. Cada bloque contiene un registro de transacciones y está sellado por un hash criptográfico. Una de sus principales características es que es inmutable; una vez que los datos se registran en la cadena, no se pueden alterar sin cambiar todos los bloques anteriores, lo que otorga seguridad y transparencia al sistema.

Desde su concepción con la aparición del Bitcoin en 2009, el uso de blockchain ha trascendido las criptomonedas. Hoy en día, diversos sectores como el financiero, sanitario, logístico y gubernamental están explorando y adoptando blockchain para rastrear cadenas de suministro, validar identidades, registrar propiedades y mucho más.

Esto ofrece una plataforma transparente y segura que permite a los individuos tener control total sobre su identidad digital, sin depender de entidades centralizadas. Por lo tanto, es esencial para una gestión soberana que busca devolver el control de los datos a los propios usuarios.

2.4 Autorización delegada

La autorización delegada es un proceso que permite a un usuario otorgar permisos a una entidad o aplicación para actuar en su nombre, sin necesidad de compartir credenciales o información sensible. Este mecanismo es crucial en entornos donde es necesario dar acceso temporal o limitado a ciertos recursos.

Con el auge de las aplicaciones y plataformas digitales, la autorización delegada ha cobrado relevancia. Protocolos como OAuth2.0 se han establecido como estándares en la industria para gestionar estos permisos, permitiendo que aplicaciones terceras accedan a información o realicen acciones sin comprometer la seguridad del usuario.

Actualmente, se ha centralizado la autorización delegada en las principales corporaciones tecnológicas, Google, Microsoft y Apple, la cuales, poseen los datos históricos de los accesos a cualquier servicio que se apoye en la autenticación delegada de estos (actualmente la gran mayoría). Esto permite una mayor facilidad de creación de cuentas y acceso a nuevos servicios a costa de una gran pérdida de privacidad.

La combinación de blockchain y autorización delegada potencia la seguridad y la privacidad. Con blockchain, se puede verificar y registrar cualquier delegación de autoridad, mientras que Web 3.0, con su enfoque descentralizado, garantiza que la gestión de identidades y autorizaciones sea más transparente y menos susceptible a puntos de fallo únicos.

2.5 Conclusión

Se ha navegado por la digitalización, abordando la evolución histórica y los problemas inherentes de la pérdida de privacidad y control. La introducción de tecnologías como blockchain y mecanismos de autorización delegada se presentan como soluciones prometedoras para restablecer la soberanía individual en el mundo digital.

Aunque se han realizado avances significativos, todavía queda mucho camino por recorrer. La adopción generalizada de blockchain y Web 3.0, así como la mejora continua de los protocolos de autorización, determinarán cómo evolucionamos hacia un futuro digital más transparente y controlado por el usuario.

En el contexto de estos desafíos y oportunidades, este proyecto destaca como un paso adelante hacia un modelo donde los ciudadanos tienen el control, la transparencia y la seguridad en la gestión de su identidad y datos, subrayando su relevancia en el panorama actual.

3 Definición del Trabajo

3.1 Justificación

En el contexto actual de una digitalización completa de la mayor parte de los procesos de nuestro día a día, vemos como hay sectores más adelantados que otros a abrazar esta filosofía. En concreto nuestra administración pública, el destino principal (aunque no necesariamente el único) de este proyecto, tiene varios problemas para adaptarse al ritmo que el resto de la industria.

En un mundo donde la digitalización ha permeado casi todos los aspectos de nuestras vidas, encontramos que ciertos sectores aún luchan por mantenerse al ritmo del progreso tecnológico. La administración pública, el principal foco de este proyecto, no está exenta de estos desafíos.

La vastedad y complejidad inherentes a organismos de gran envergadura, como la administración pública, conllevan procesos intrincados y a menudo engorrosos. Aunque en teoría muchos de estos procesos han sido digitalizados, la realidad muestra que aún subsisten procedimientos manuales. Si bien se pueden llevar a cabo digitalmente mediante certificados digitales, a menudo este paso digital enmascara una estructura operativa anacrónica detrás de las cortinas.

La digitalización trae consigo otro dilema, uno que recae directamente sobre los hombros del ciudadano: la privacidad y la transparencia. Vivimos en una era donde la línea entre la privacidad y la publicidad es borrosa, con conglomerados tecnológicos que se alimentan de nuestros datos para mantener sus ecosistemas.

Ante este conflicto entre la automatización y la privacidad, surge este proyecto, apuntando a blockchain como el pilar tecnológico para restaurar la confianza. Blockchain no solo asegura la integridad y propiedad de la información, sino que también promete una transparencia sin precedentes. Sumado a ello, mediante el desarrollo de software específico, se presenta una solución refinada para automatizar transacciones entre entidades públicas y bancarias.

No obstante, más allá de la eficiencia, este proyecto reconoce la autonomía del individuo, ofreciendo un sistema de autorizaciones que otorga al ciudadano el poder de decidir cuándo y cómo se accede a sus activos. Esta temporalidad y selectividad en los permisos garantizan un equilibrio entre autonomía y automatización.

Y para consolidar el compromiso con la privacidad, el proyecto adopta una autenticación web3.0. Esta metodología permite al usuario acceder mediante una “Wallet” de Ethereum, asegurando que la propiedad y control de la información permanezcan en manos del titular.
En resumen, este proyecto surge como respuesta a un imperativo moderno: diseñar sistemas que, mientras maximizan la eficiencia y automatización, no sacrifiquen la transparencia, la integridad y, sobre todo, la privacidad del usuario.

3.2 objetivos

Las necesidades actuales van totalmente de la mano de los objetivos, los cuáles se traducen en objetivos generales y objetivos específicos.

3.2.1 Objetivos generales

Los objetivos generales buscan guiar el proyecto y establecer la estructura claro de lo que se quiere lograr. Principalmente, este proyecto se ha centrado en:

Mejorar la experiencia del usuario: Uno de los principales objetivos de la autorización delegada es simplificar la experiencia del usuario al permitirles acceder y utilizar servicios y aplicaciones de manera más conveniente. Esto se logra eliminando la necesidad de crear y recordar múltiples credenciales de inicio de sesión y proporcionando un flujo de autenticación más fluido y sin fricciones.

Aumentar la seguridad y el control de acceso: La autorización delegada busca mejorar la seguridad al permitir a los usuarios otorgar permisos específicos a aplicaciones y servicios sin compartir sus credenciales principales. Esto reduce la exposición de las credenciales de usuario y brinda a los propietarios de los recursos un mayor control sobre cómo se utilizan sus datos.

Facilitar la interoperabilidad entre diferentes sistemas y aplicaciones. Al adoptar estándares comunes de autorización delegada (web3.0, blockchain, …), se pueden crear integraciones más sencillas y permitir a los usuarios acceder y compartir datos entre plataformas de manera segura y confiable.

Mejorar la eficiencia y escalabilidad: se busca mejorar la eficiencia al permitir que los servicios y aplicaciones accedan a recursos protegidos en nombre del usuario, sin tener que autenticarse y autorizarse por separado en cada interacción. Esto puede reducir la carga en los servidores y mejorar el rendimiento general del sistema.

Promover la adopción de estándares: Al implementar proyectos de autorización delegada que sigan estándares reconocidos, se busca promover la adopción generalizada de esos estándares en la industria. Esto crea un ecosistema más coherente y facilita la integración y la colaboración entre diferentes servicios y plataformas.

Gestionar la delegación de permisos permitiendo al usuario definir la duración máxima de un permiso para evitar que sean, por defecto en muchas aplicaciones actuales, indefinido. Además, crear un sistema que permita inhabilitar cualquier permiso en tiempo real, en cualquier momento y con efectividad instantánea.

3.2.2 Objetivos específicos

Los objetivos específicos complementan a los generales, detallando las acciones llevadas a cabo para cumplirlos:

Integración con Blockchain: Utilizar la tecnología blockchain para garantizar la integridad, propiedad y transparencia de la información en las transacciones.

Implementación de Autenticación Web3.0: Integrar un sistema de autenticación basado en Web3.0 que permita a los usuarios acceder al sistema mediante una Wallet de Ethereum, asegurando así la propiedad y control sobre sus datos.

Sistema de Autorizaciones: Desarrollar un mecanismo que permita a los ciudadanos gestionar permisos sobre sus activos y transacciones, otorgando o revocando el acceso a las entidades públicas según lo deseen.

Interfaz de Usuario Amigable: Crear una plataforma o panel de usuario fácil de utilizar que centralice todas las gestiones y transacciones, minimizando la necesidad de visitar múltiples plataformas o entidades gubernamentales.

Educación y Sensibilización: Fomentar la educación y conciencia sobre la importancia de la privacidad, transparencia y la gestión soberana de la información en el contexto digital actual.

Colaboración Interinstitucional: Establecer alianzas y colaboraciones con entidades públicas, instituciones financieras y proveedores de servicios tecnológicos para garantizar la implementación exitosa y el soporte del sistema.

3.3 Metodología

Para este proyecto, se ha hecho un análisis entre las metodologías más comunes de la actualidad:

- **Waterfall o cascada**: es un enfoque secuencial y lineal donde las fases del proyecto se llevan a cabo en un orden predeterminado, pasando de una fase a la siguiente una vez que se completa la anterior. En el caso de un proyecto de autorización delegada, esto implicaría seguir un flujo desde la planificación y diseño, hasta la implementación y pruebas, y finalmente la entrega y el mantenimiento.

- **Agile o ágil**: se centra en la flexibilidad y en la entrega iterativa e incremental. En lugar de tener una planificación y diseño detallados al comienzo del proyecto, se trabaja en ciclos cortos llamados sprints, donde se priorizan y desarrollan las características clave del sistema. Esto permite una adaptación más rápida a medida que se aprende y se obtiene retroalimentación a lo largo del proceso de desarrollo.

- **Test-Driven development (TDD) o desarrollo impulsado por pruebas**: es una metodología en la que las pruebas se escriben antes de escribir el código fuente. En el contexto de un proyecto de autorización delegada, esto implicaría definir primero los casos de prueba y los criterios de aceptación, y luego desarrollar el sistema para que pase las pruebas. Esta metodología ayuda a garantizar que el sistema cumpla con los requisitos y funcione correctamente.

- **Desarrollo basado en prototipos**: Se centra en la creación rápida de prototipos funcionales del sistema para obtener retroalimentación temprana y validación de los requisitos. En el caso de un proyecto de autorización delegada, se podrían crear prototipos para mostrar las interacciones de autenticación y autorización y recibir comentarios de los usuarios antes de implementar la solución final.

- **Desarrollo Lean**: El desarrollo Lean se basa en el principio de minimizar el desperdicio y maximizar el valor entregado. Se centra en la identificación y eliminación de actividades que no agregan valor y en la optimización del flujo de trabajo. En el contexto de un proyecto de autorización delegada, se buscaría reducir la complejidad y mejorar la eficiencia del proceso de autorización.

Muchas de ellas podrían ser complementarias entre sí, en cualquier caso, para elegir la más correcta se ha optado por ir descartando las que menos encajaban. A continuación, se detallan los motivos de descarte y las metodologías descartadas.

- **Ausencia de cliente final:** Al no existir la retroalimentación con un cliente final, la metodología basada en prototipos carece de sentido.

- **Falta de definición de todos los detalles del producto:** Al ser un proyecto que ha requerido mucha investigación y no ha tenido un roadmap claro desde el principio (dada la necesidad de realizar distintas pruebas con distintas tecnologías y métodos), se descartan las metodologías de cascada (dado su enfoque de fases lineal) y, por último, la metodología TDD, la cual requiere que se escriban las pruebas teniendo clara la funcionalidad final.

Tras el descarte, quedan las metodologías “Agile” y “Lean”. Las cuáles se optó utilizar de forma complementaria.
La metodología Agile se centra en la entrega continua de tareas, las cuáles, debe ser muy atómicas y modulares. Para esta labor, de creó un panel de tipo “Kanban” para el desglose y planificación de tareas.

3.4 Planificación y Estimación Económica

El proyecto se llevará a cabo en varias etapas, cada una de ellas con actividades específicas y entregables clave.

Se han detallado las etapas correspondientes al desarrollo del TFG (las tres primeras) y un desglose hipotético de la implementación de este proyecto en el mercado (última etapa).

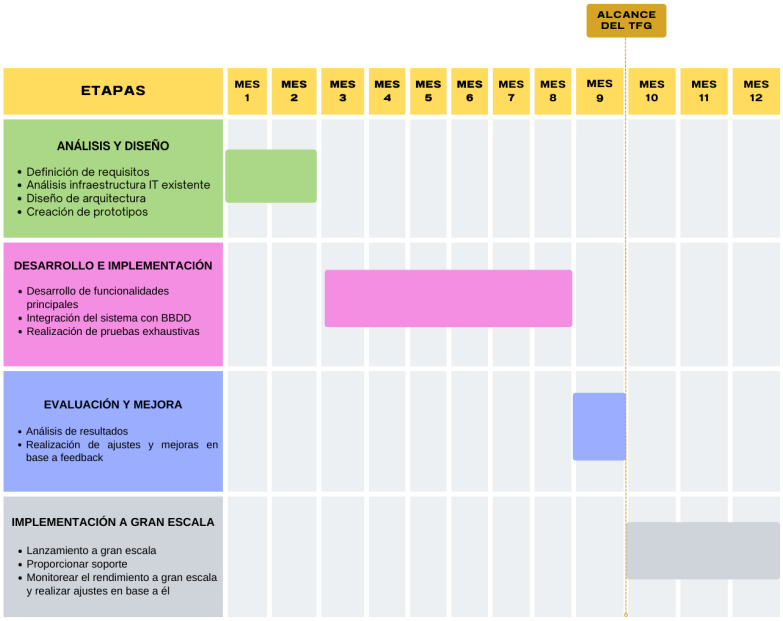


Figura 1. Diagrama de Gantt detallando la planificación del proyecto

Aunque la mayor parte del esfuerzo del proyecto se ha invertido en la implementación de la aplicación, se ha tenido que hacer mucho trabajo de investigación para poder llevarlo a cabo, tanto a nivel conceptual como técnico.

A continuación, se detalla cada una de las etapas del proyecto:

3.4.1 Etapa 1: Análisis y Diseño (Duración: 2 meses)

Definir los requisitos del sistema en colaboración con los stakeholders y las entidades gubernamentales involucradas.
Realizar un análisis detallado de la infraestructura tecnológica existente y evaluar la viabilidad de la migración a una red blockchain en el futuro.

Diseñar la arquitectura del sistema distribuido, incluyendo la integración con Metamask para la autenticación y la posible integración con una red blockchain.
Crear prototipos y maquetas para validar el diseño y la experiencia del usuario.

3.4.2 Etapa 2: Desarrollo e Implementación (Duración: 6 meses)

Desarrollar las funcionalidades principales del portal, incluyendo el sistema de autenticación con Metamask, la gestión de permisos y cuentas bancarias, y la automatización de pagos.

Integrar el sistema con las bases de datos existentes y establecer conexiones seguras con las entidades gubernamentales para el intercambio de información.
Realizar pruebas exhaustivas para garantizar la seguridad, la funcionalidad y la usabilidad del sistema.

Llevar a cabo la implementación piloto del sistema con un grupo reducido de usuarios para obtener retroalimentación y realizar mejoras.

3.4.3 Etapa 3: Evaluación y Mejora (Duración: 1 mes)

Analizar los resultados y la retroalimentación obtenida durante la fase piloto.

Realizar ajustes y mejoras en el sistema en base a los comentarios de los usuarios y los hallazgos del análisis.

Preparar la infraestructura y los recursos necesarios para la implementación a gran escala.

3.4.4 Etapa 4: Implementación a Gran Escala (Duración: 3 meses)

Lanzar el sistema a gran escala y ampliar gradualmente el número de usuarios.

Proporcionar capacitación y soporte a las entidades gubernamentales y a los ciudadanos para garantizar una transición sin problemas hacia el nuevo sistema.

Monitorear y evaluar el rendimiento del sistema durante el proceso de implementación a gran escala y realizar ajustes adicionales si es necesario.

3.4.5 Estimación del coste de desarrollo

La estimación del coste de desarrollo del proyecto se basará en varios factores, como la complejidad del sistema, la duración del proyecto y los recursos necesarios para llevar a cabo cada etapa. Se espera que el coste total del desarrollo sea de aproximadamente de unos 562.000€, que incluye gastos en:

Recursos Humanos: Los recursos humanos son una parte crucial del proyecto, ya que el éxito de su implementación depende en gran medida del talento y la experiencia del equipo de desarrollo. Se requerirá un equipo multidisciplinario que incluya ingenieros de software, arquitectos de sistemas, especialistas en blockchain, diseñadores y gestores de proyectos. La estimación del coste de recursos humanos dependerá de la duración del proyecto y las tarifas salariales del equipo. Teniendo en cuenta la dificultad de atraer talento y el tamaño inicial de la empresa, se externalizaría esta función, asumiendo un coste anual de unos 20.000€.

Tecnología y Herramientas: La implementación del sistema distribuido requerirá el uso de tecnologías y herramientas específicas. Esto incluirá licencias de software, infraestructura en la nube y herramientas de desarrollo. Se deberán considerar los costes iniciales y los costes continuos de mantenimiento y actualización de estas tecnologías.

Inicialmente, se están utilizando tecnologías libres, las cuales no tienen ningún coste. Se tendría que adelantar solo el coste de la infraestructura y mantenimiento en nube. Suponiendo que la solución funcionara y tuviera un volumen de usuarios elevado, unos 1.000€ máximo al mes en infraestructura.

Capacitación y Soporte: Comprende los gastos asociados a la capacitación del personal de las entidades gubernamentales y los usuarios finales para asegurar una correcta adopción del sistema. Además, se debe proporcionar un soporte técnico adecuado durante el proceso de implementación y de forma continua posteriormente. Estos costes pueden variar según el número de usuarios y la complejidad de la formación requerida, pero se estima que estarían alrededor de los 2.000€ anuales en la compra de cursos y certificaciones para el equipo de técnico.

Personal técnico: Inicialmente, la mayor inversión de la aplicación hasta poder crecer y cambiar la estrategia. Inicialmente, se terminar la aplicación y llevar a sus máximas capacidades en 2 años de desarrollo con un equipo de 4 desarrolladores con un sueldo medio de 50.000€ anuales (con impuestos simulado a 65.000€ de gastos por individuo).

Otros Gastos: Además de los costes mencionados anteriormente, también se deben tener en cuenta otros gastos, como licencias adicionales, gastos de viaje, costes legales y gastos de marketing. Estos gastos pueden ser variables según las circunstancias específicas del proyecto. Una estimación razonable para los otros gastos sería de 10.000€.

Es importante destacar que esta estimación es una aproximación y que los costes reales pueden variar según la complejidad y los requerimientos específicos del proyecto. Se realizará un seguimiento periódico del presupuesto para asegurar la correcta gestión de los recursos durante el desarrollo del proyecto.

3.4.6 Viabilidad económica

La viabilidad económica del proyecto se basa en la estimación de los costes y los beneficios esperados. Para evaluar su viabilidad, es esencial comparar los costes proyectados con los beneficios esperados a lo largo del tiempo.

Los beneficios del proyecto pueden manifestarse en varias formas, como la reducción de costes operativos y administrativos para las entidades gubernamentales, la mejora de la experiencia del usuario y la posibilidad de ampliar el servicio a un mayor número de contribuyentes.

La generación de ingresos para el proyecto puede provenir de diversas fuentes, como tarifas de servicio para las entidades gubernamentales o incluso un modelo de suscripción para los usuarios finales.

En cualquier caso, sabiendo que es complicado generar ingresos en los primeros años de vida de un proyecto, inicialmente, se buscaría inversión externa para poder acometer el desarrollo completo de la solución.

4 Descripción de las Tecnologías

El proyecto se estructura en base a los siguientes componentes generales, los cuales, permiten el correcto funcionamiento de la plataforma.

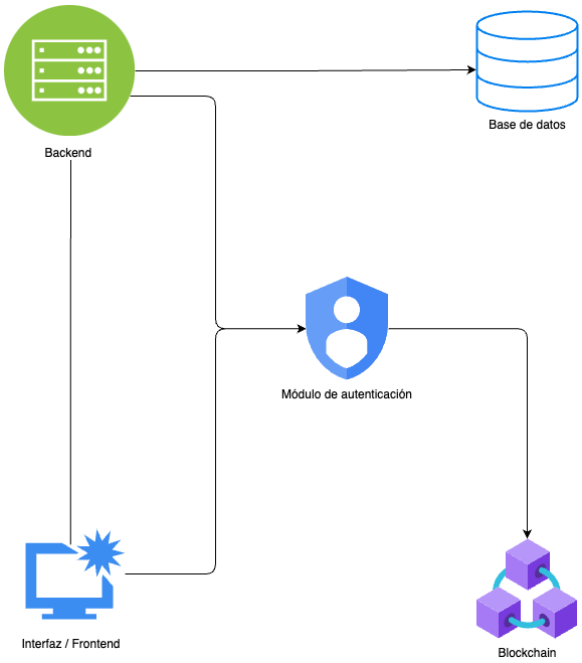


Figura 2. Detalle de la arquitectura global

Se ha optado por un diseño modular, separando claramente cada una de las piezas lógicas de la aplicación. Esto permite trabajar de forma atómica sobre la aplicación, facilitando crear cambios en cada componente.

A continuación, se detalla una asociación de las tecnologías que se usan por cada uno de los componentes de la aplicación.

- **Módulo de autenticación**: Metamask.
- **Backend**: Flask (Python).
- **Frontend**: HTML, CSS y Javascript.
- **Blockchain**: Ethereum.
- **Base de datos**: SQLite y MySQL.

4.1 Módulo de autenticación

Este componente es el puente entre la aplicación "tradicional" y la redes blockchain. Buscar automatizar la gestión de las sesiones de los usuarios, credenciales y delegaciones sobre una cadena de bloques.

Concretamente, permite que la aplicación consulte sobre un servicio de terceros el cual, a su vez, valida la información en una cadena de bloques. Esto permite, por una parte, mejorar la confianza de la aplicación al utilizar un servicio estándar, conocido y maduro en el ámbito; por otra parte, facilita la integración de la aplicación con una cadena de bloques al abstraer al usuario de la parte programática de consultar a la cadena de bloques.

4.1.1 Metamask

En las primeras fases de desarrollo del Sistema Distribuido, se ha realizado una cuidadosa evaluación de las opciones disponibles para el sistema de autenticación al portal a utilizar. Después de considerar varias alternativas, se ha elegido integrar Metamask como el sistema de autenticación preferido. A continuación, se presentan las razones detrás de esta elección y las ventajas que Metamask ofrece en comparación con otras opciones consideradas.

Metamask es ampliamente reconocido en el ecosistema de criptomonedas y las aplicaciones descentralizadas (dApps) basadas en Ethereum. Su adopción masiva y la existencia de una comunidad activa de usuarios y desarrolladores brindan una gran confianza en su seguridad y funcionalidad. Esta amplia adopción también facilita la integración con otras aplicaciones y servicios basados en Ethereum.
Una de las ventajas clave de Metamask es su interoperabilidad. No solo es compatible con Ethereum, sino que también admite otras redes y estándares basados en Ethereum, como Binance Smart Chain y Polygon. Esta interoperabilidad garantiza que los usuarios puedan acceder al portal desde diferentes plataformas y aprovechar las ventajas de varias redes blockchain, lo que proporciona flexibilidad en la elección de la red adecuada para el proyecto.

En términos de seguridad y privacidad, Metamask utiliza técnicas criptográficas sólidas para proteger las claves privadas de los usuarios. Las claves privadas nunca abandonan el dispositivo del usuario y se almacenan de forma segura en la billetera de Metamask. Esta arquitectura de seguridad proporciona un nivel adicional de protección de datos personales y evita posibles vulnerabilidades en la autenticación.

Además, Metamask permite a los usuarios revisar y aprobar cada transacción antes de que se lleve a cabo. Esto brinda un mayor control a los usuarios y previene posibles fraudes o transacciones no deseadas. La capacidad de revisar y aprobar transacciones garantiza una mayor transparencia y confianza en el proceso de autenticación.

En términos de experiencia de usuario, Metamask ofrece una interfaz intuitiva y fácil de usar. Los usuarios pueden crear y gestionar sus carteras de criptomonedas de manera sencilla. La integración de Metamask en el portal proporciona una experiencia de

autenticación simplificada y sin complicaciones para los usuarios. Además, la interfaz de Metamask permite a los usuarios controlar y administrar sus cuentas, ver el historial de transacciones y administrar los permisos de acceso de forma intuitiva.

En comparación con otras opciones consideradas, como web3auth, Metamask se destaca por su amplia adopción y soporte comunitario. La existencia de una gran base de usuarios y una comunidad activa de desarrolladores garantiza que el sistema de autenticación esté respaldado por una sólida infraestructura y se mantenga actualizado con las últimas tendencias y mejoras de seguridad.

Además, Metamask ofrece una mayor interoperabilidad al ser compatible con múltiples redes blockchain. Esto brinda a los usuarios la posibilidad de elegir la red adecuada para sus necesidades y permite una mayor flexibilidad en la implementación del sistema distribuido.

4.2 Backend

Aquí se aborda la parte de la aplicación que se encarga de manejar la lógica, el procesamiento de datos y la comunicación con la base de datos. El backend es esencial para garantizar el funcionamiento adecuado de la aplicación y para manejar todas las interacciones entre el frontend y la base de datos.
En un sistema de este tipo, el backend estaría construido en Flask, un framework de Python que permite desarrollar aplicaciones web de manera eficiente. Flask utiliza el concepto de rutas y vistas para gestionar las solicitudes entrantes y las respuestas salientes.

4.2.1 Python Flask

Para la parte lógica y programática de la aplicación se ha optado por utilizar Python. Este lenguaje tiene una sintaxis muy limpia y sencilla, el cual, resulta muy agradable de programar.

Flask nos facilita la construcción de un API o servicio web, marcando unas guías de estructura de ficheros, aportando unas librerías que facilitan el enrutado de los distintos tipos de peticiones web (GET, POST, PUT… etc.) y, junto con SQLAlchemy (se detallará más adelante), un sistema ORM, que nos facilitará la comunicación con diversos motores de base de datos.
Dado el alcance y tamaño del proyecto se optó por utilizar flask. Entre los tres, el que menor curva de aprendizaje tiene, aunque posiblemente, el más complicado de escalar a largo plazo.

Flask es un framework web ligero y flexible para el lenguaje de programación Python. Fue creado por Armin Ronacher y lanzado por primera vez en 2010.

A diferencia de otros frameworks web de Python, como Django, Flask no impone una estructura de directorios o un conjunto particular de herramientas. En su lugar, Flask permite a los desarrolladores tener un mayor control y flexibilidad en la organización de su aplicación web.

Flask se basa en el concepto de "microframework", lo que significa que proporciona solo las características esenciales para el desarrollo web, como enrutamiento URL, gestión de solicitudes y respuestas, y soporte para cookies. Sin embargo, Flask es altamente extensible y se pueden agregar muchas funciones adicionales mediante extensiones.
Debido a su enfoque minimalista y su simplicidad, Flask es muy popular entre los desarrolladores de Python que desean construir aplicaciones web rápidas y livianas. Motivos principales por los que se optó por esta tecnología.

4.3 Frontend

La interfaz de usuario, comúnmente llamada frontend, es la parte visible y con la que los usuarios interactúan en una aplicación o sitio web. Es la capa que permite a los usuarios ver, navegar y realizar acciones dentro de la aplicación. Su objetivo principal es proporcionar una experiencia de usuario efectiva, intuitiva y atractiva.
La elección de utilizar HTML5, CSS y JavaScript como tecnologías para desarrollar el frontend de tu aplicación tiene dos razones clave:

- Son un estándar en la industria de servicios web.
- Separas la responsabilidad: HTLM se utiliza para estructurar el contenido; CSS del diseño; y Javascript de la interactividad y lógica de la aplicación.

4.3.1 HTML 5

HTML5 (HyperText Markup Language 5) es la última versión del lenguaje de marcado estándar utilizado para estructurar y presentar contenido en la web. HTML5 introduce una serie de nuevas características y elementos semánticos que permiten a los desarrolladores crear sitios web más ricos y sofisticados. Estos elementos incluyen encabezados, párrafos, enlaces, imágenes, tablas, formularios y muchos otros. HTML5 también ofrece soporte para video y audio integrados, gráficos vectoriales escalables (SVG), canvas para gráficos dinámicos y más. Proporciona una base sólida para construir la estructura de una página web y establecer la semántica correcta para el contenido.

4.3.2 CSS

CSS (Cascading Style Sheets) es un lenguaje de estilo utilizado para definir la apariencia y el diseño de los elementos en un documento HTML. Con CSS, los desarrolladores pueden aplicar reglas de estilo a los elementos HTML, como colores, fuentes, márgenes, tamaños, efectos de sombra y más. CSS proporciona una forma de separar el contenido y la presentación en un sitio web, lo que permite un mayor control y flexibilidad en la personalización del aspecto visual. Además, CSS ofrece la capacidad de aplicar estilos de forma selectiva a través de selectores y ofrece la posibilidad de crear diseños responsivos para adaptarse a diferentes dispositivos y tamaños de pantalla.

4.3.3 Javascript

JavaScript es un lenguaje de programación interpretado que se utiliza para crear interactividad en los sitios web. Es un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador web del usuario y permite realizar acciones y cambios dinámicos en una página web sin tener que recargarla por completo. JavaScript se utiliza para manipular elementos HTML, responder a eventos del usuario, realizar validaciones de formularios, animaciones, llamadas a servicios web, almacenamiento de datos en el navegador y mucho más. Es una tecnología fundamental para la creación de aplicaciones web interactivas y dinámicas.

En resumen, HTML5 proporciona la estructura y la semántica del contenido web, CSS se encarga del estilo y el diseño visual, y JavaScript añade interactividad y funcionalidad dinámica a los sitios web. Estas tres tecnologías se complementan entre sí y son fundamentales para el desarrollo moderno de páginas web.
En este proyecto, se ha utilizado HTML y CSS para construir todas las plantillas, vistas e interfaces de la aplicación. Por otra parte, javascript permite gestionar las sesiones de login con Metamask e intercomunicar algunas peticiones al backend (Flask) para obtener información de forma dinámica sin recargar la web.

4.4 Cadena de bloques

Una cadena de bloques, o blockchain en inglés, es una tecnología de registro descentralizado y distribuido que se utiliza para almacenar y gestionar datos de manera segura, transparente e inmutable. La característica distintiva de una cadena de bloques es su estructura en forma de cadena, donde los datos se agrupan en bloques y se conectan secuencialmente. Cada bloque contiene un conjunto de transacciones u otros tipos de información, y cada bloque está vinculado al bloque anterior a través de una función criptográfica llamada "hash".

En la aplicación de gestión soberana, la tecnología blockchain, o cadena de bloques, tendría varios roles clave que contribuirían a mejorar la seguridad, la transparencia y la confianza en el sistema garantizando la seguridad, la transparencia y la confianza en

todas las interacciones relacionadas con los permisos y los cargos. Proporcionaría un registro seguro y verificable de todas las acciones realizadas en la plataforma, lo que mejoraría la experiencia de los usuarios y las entidades, al tiempo que brindaría una mayor integridad de los datos y registros.

Dentro de las principales cadenas de bloques que existen en la actualidad, Ethereum es una de las más eficientes, ecológicas, adoptadas y flexibles, por lo que se consideró idónea para este proyecto.

4.4.1 Ethereum

Ethereum es una plataforma descentralizada y una criptomoneda que permite la ejecución de contratos inteligentes. Fue propuesto por Vitalik Buterin en 2013 y lanzado en 2015. A diferencia de Bitcoin, que se centra en ser una moneda digital, Ethereum tiene como objetivo ser una plataforma para construir aplicaciones descentralizadas (dApps) y contratos inteligentes.

En Ethereum, los contratos inteligentes son programas informáticos autónomos que se ejecutan en la blockchain de Ethereum. Estos contratos inteligentes contienen la lógica y las reglas acordadas por las partes involucradas en una transacción, eliminando la necesidad de intermediarios y permitiendo transacciones directas y seguras entre las partes. Los contratos inteligentes son escritos en lenguaje de programación Turing completo llamado Solidity.

La criptomoneda nativa de Ethereum se llama Ether (ETH) y se utiliza para pagar las tarifas de transacción y recompensar a los mineros que validan las transacciones y mantienen la seguridad de la red (PoW, Proof-of-Work). Este sistema cambió recientemente a Proof-of-stake (PoS), este es un sistema por lotería que requiere que los participantes de la red compren y bloqueen los tokens nativos de un protocolo en un contrato inteligente. Los participantes que hacen staking con tokens tienen la oportunidad de ser seleccionados al azar para proponer nuevos bloques.

Tres años después del lanzamiento del Bitcoin, dos desarrolladores llamados Scott Nadal y Sunny King desarrollaron el mecanismo de consenso de PoS para abordar las deficiencias energéticas que plantea el sistema de proof-of-work.

Cuanto mayor sea el número de tokens con los que haga staking una persona, mayor será su probabilidad de ser seleccionada. Piense en esto como comprar varios billetes de lotería. Aumentarán sus probabilidades cuantos más billetes tenga, pero eso no garantiza que gane. Una persona que haya comprado un billete solamente o, en este caso, que haya hecho staking con la cantidad mínima de tokens, puede ser seleccionada en su lugar de todos modos.

Esta idea se sirve del mismo principio que PoW al exigir a los validadores que inviertan su propio dinero, pero elimina la necesidad de operar máquinas especializadas y de alto consumo energético.

Una de las características más destacadas de Ethereum es su capacidad para soportar la creación de tokens personalizados y la realización de Ofertas Iniciales de Moneda (ICO).

Esto ha permitido el surgimiento de un ecosistema amplio de proyectos construidos sobre la plataforma de Ethereum, incluyendo aplicaciones financieras descentralizadas (DeFi), juegos blockchain, sistemas de votación y mucho más.

4.5 Bases de datos

Las bases de datos son sistemas organizados y estructurados para almacenar, gestionar y recuperar información de manera eficiente.

En el contexto de la informática y el software, una base de datos se utiliza para almacenar datos de manera que puedan ser accesibles, actualizables y manipulables de manera efectiva. Principalmente se utilizan bases de datos de dos tipos:

- **Bases de datos relacionales (DBMS).**
- **Bases de datos no relacionales.**

La base de datos relacional ofrece una forma organizada y estructurada de almacenar datos, lo que facilita la gestión y recuperación de información. Es ampliamente utilizada en aplicaciones empresariales, sistemas de administración y muchas otras aplicaciones donde la integridad y la consistencia de los datos son esenciales.

Las bases de datos no relacionales permiten almacenar información de forma desestructurada, facilitando la colecta de información muy heterogénea sin miedo a generar errores de integridad.

Teniendo en cuenta las necesidades intrínsecas de integridad y consistencia de datos de la aplicación, una base de datos relaciona se considera lo más idóneo para la aplicación. Estas bases de datos, utiliza un lenguaje llamado SQL para realizar sus operaciones.

Por último, se han utilizado dos tecnologías SQL: SQLite y MySQL. El primero en durante la fase de desarrollo (detallado en el capítulo 6, entorno de desarrollo), al no necesitar un servidor, mientras que el segundo en entornos productivos.

4.5.1 SQL

SQL (Structured Query Language) es un lenguaje de programación utilizado para administrar y manipular bases de datos relacionales. Es el estándar de facto para interactuar con sistemas de gestión de bases de datos relacionales (RDBMS).

Permite realizar diversas operaciones en una base de datos, como consultar, insertar, actualizar y eliminar datos. Con SQL, se pueden crear, modificar y eliminar tablas, así como definir restricciones, índices y relaciones entre las tablas.

Es ampliamente utilizado en el desarrollo de aplicaciones web y empresariales que requieren almacenamiento y recuperación de datos. Los sistemas de gestión de bases de datos populares, como MySQL, PostgreSQL, Oracle Database y Microsoft SQL Server, son compatibles con SQL.

Para este proyecto, se ha utilizado SQLAlchemy, el cual, permite operar de una forma transparente entre múltiples motores de bases de datos. Durante la fase de desarrollo, se ha trabajado con SQLite, aunque puede ser sustituido por PostgreSQL o MySQL en un entorno productivo sin ninguna modificación.

4.5.2 SQLite

SQLite es un DBMS ligero y autónomo que se destaca por su simplicidad y portabilidad. Es una base de datos embebida, lo que significa que el motor de base de datos se integra directamente en la aplicación que lo utiliza, sin necesidad de un servidor de base de datos separado. Algunas características clave de SQLite son:

- Sin Servidor: No requiere un servidor externo. La base de datos se almacena en un archivo local, lo que facilita su portabilidad y distribución junto con la aplicación.
- Autocontenido: La base de datos SQLite se guarda en un solo archivo, lo que la hace fácil de respaldar, migrar y distribuir.

- Ligero: Adecuado para aplicaciones que no requieren una alta concurrencia y rendimiento extremo. Ideal para aplicaciones de escritorio, móviles y en sistemas integrados.
- Transaccional: Soporta transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) para garantizar la integridad de los datos.

- SQL Completo: Ofrece soporte para consultas SQL completas, lo que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en la base de datos.

- No Requiere Configuración: No necesita una configuración compleja ni administración del servidor.

4.5.3 MySQL

MySQL es un DBMS de código abierto que se utiliza en muchos entornos empresariales y web. A diferencia de SQLite, MySQL requiere un servidor de base de datos para funcionar y admite la gestión de bases de datos a gran escala. Algunas características clave de MySQL son:

- **Cliente-Servidor:** Funciona en un modelo cliente-servidor, donde el servidor maneja las solicitudes de múltiples clientes y administra la base de datos.

- **Alto Rendimiento:** Diseñado para manejar grandes cantidades de datos y alta concurrencia. Adecuado para aplicaciones web y empresariales que requieren una gran escalabilidad.

- **Multihilo:** MySQL puede administrar múltiples conexiones y solicitudes concurrentes a través de hilos.

- **Soporte para Lenguajes de Programación:** Admite varios lenguajes de programación y frameworks a través de controladores y APIs.

- **Gestión de Usuarios y Permisos:** Ofrece un sistema robusto de gestión de usuarios y permisos para controlar el acceso a la base de datos.

- **Replicación y Alta Disponibilidad:** MySQL proporciona opciones para replicación y alta disponibilidad para garantizar la continuidad del servicio.

- **Optimización de Consultas:** Ofrece herramientas para optimizar consultas y mejorar el rendimiento de la base de datos.

5 Sistema desarrollado

El proyecto consta del uso combinado de varias tecnologías descritas en el capítulo 4. Para este sistema se necesita una infraestructura flexible y garantizar al máximo la disponibilidad y la seguridad.

Para ello, el sistema (y su conjunto de componentes y servicios) se agrupa en la caja de la derecha de la siguiente figura.

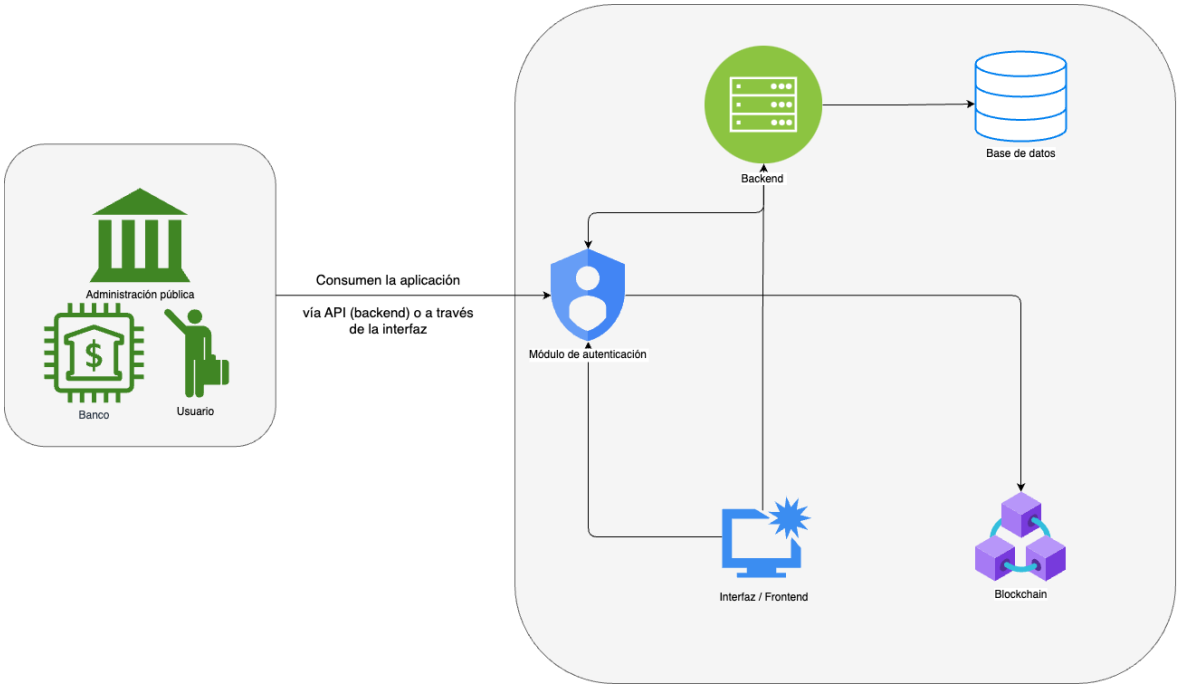


Figura 3. Diagrama de la arquitectura global y sus agentes

La solución se consume de forma externa por tres agentes principales, la administración pública, bancos y el usuario final.

En líneas generales, el sistema permite registrar todas las operaciones (cobros) de la administración pública hacia un usuario y sus respectivas entidades financieras (bancos). Esto permite unificar y centralizar el sistema en el que operan estos tres agentes, canalizándolos a través del mismo estándar de autenticación y, como máximo propósito de este proyecto, **verificando cada operación con la previa autorización del usuario delegada en la plataforma.**

Esta solución permite, a través del módulo de autenticación, validar todas las acciones de los agentes en la solución. Una vez autenticados contra la plataforma, podrán operar contra la solución a través del backend llamando al API (vía programática) o a través de la interfaz del usuario (navegador web). **Los casos de uso de los agentes de detallan en capítulo 8.**

5.1 Arquitectura de software

El sistema está formado por una interfaz o "frontend" desarrollado en las tecnologías web HTML, CSS y Javascript. Estas se apoyan en una API que hace de "backend" y, por último, el API se apoya en Metamask, MySQL y Ethereum. El siguiente diagrama muestra la interconexión de las distintas tecnologías.

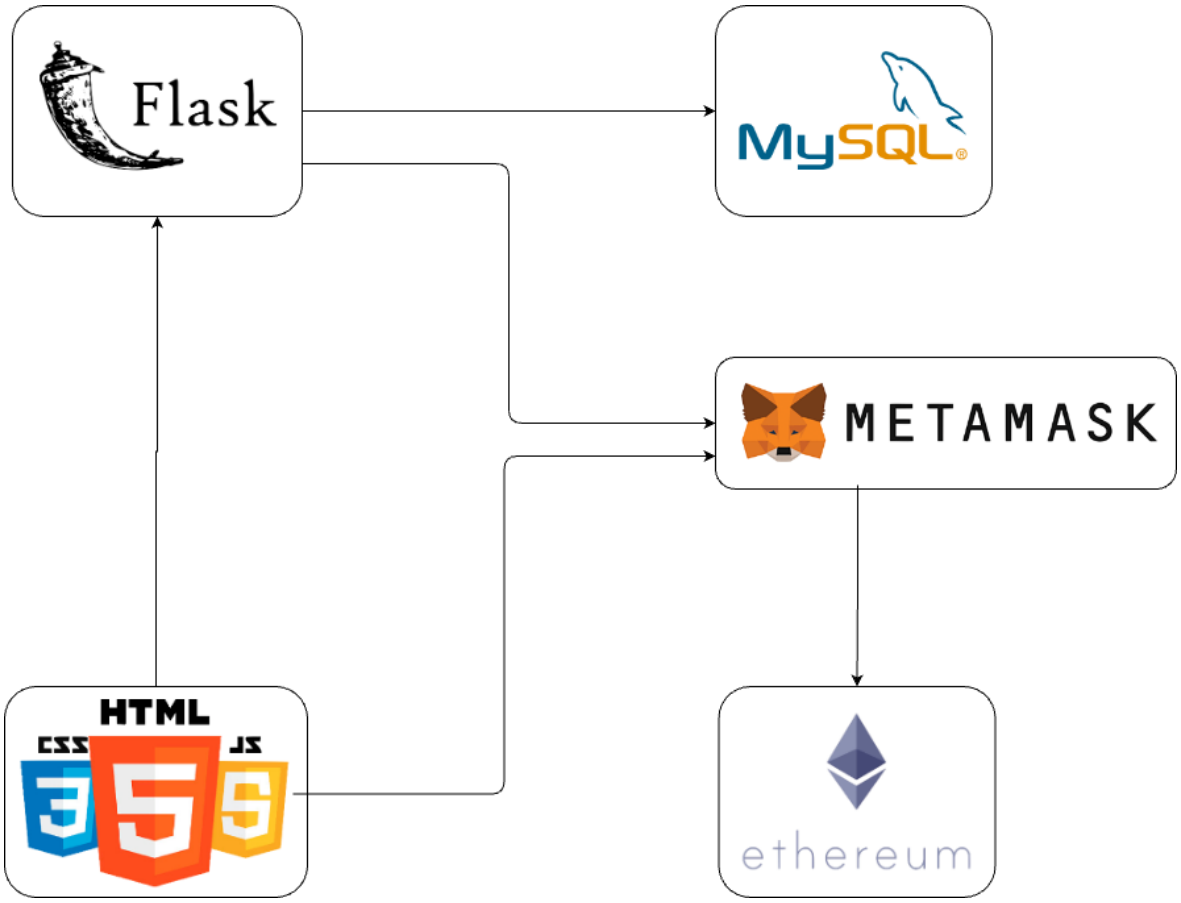


Figura 4. Arquitectura de software

Este es el esquema esperado para su funcionamiento. En cualquier caso, la arquitectura real que se ha usado para su desarrollo y pruebas de detalle en el capítulo 6, Entorno de desarrollo. También se explica los motivos y complejidades de simular un entorno donde poder hacer pruebas en local sin imputar los costes de las transferencias de Ethereum.

La arquitectura se ha realizado apoyándose en Flask como backend y pilar de la lógica de la aplicación. Aunque en el capítulo 2 ya se han descrito las tecnologías utilizadas, a continuación, se describirá los motivos de la elección de las mismas y la comunicación e interdependencias entre ellas mismas.

5.2 Motivos de elección de las tecnologías

Aunque las tecnologías que se utilizan para el proyecto se detallan en el capítulo dos, se ha querido hacer un inciso en el motivo de elección de estas para este proyecto. A continuación, se detallan los siguientes motivos:

Ethereum es una plataforma líder en la industria para la construcción de aplicaciones descentralizadas (dApps) y contratos inteligentes en blockchain. Sus motivos y puntos fuertes incluyen:

- **Smart contracts**: Ethereum permite la implementación de contratos inteligentes, que son programas autónomos y ejecutables que se ejecutan en la blockchain. Esto permite la automatización de transacciones y la implementación de lógica empresarial compleja de manera confiable y segura.

- **Interoperabilidad**: Ethereum es compatible con una amplia gama de bibliotecas, herramientas y estándares que facilitan la interoperabilidad con otras plataformas y dApps.

- **Comunidad y adopción**: Ethereum tiene una gran comunidad de desarrolladores y una amplia adopción en la industria. Esto significa que hay abundantes recursos, documentación y soporte disponibles para el desarrollo de proyectos basados en Ethereum.

Flask es un framework de desarrollo web en Python conocido por su simplicidad y flexibilidad. Algunos de sus motivos y puntos fuertes incluyen:

- Facilidad de uso: tiene una curva de aprendizaje suave y su sintaxis clara y concisa facilita el desarrollo rápido de aplicaciones web.

- Flexibilidad: permite una gran libertad en el diseño y la estructura de las aplicaciones web, lo que te permite adaptarla a tus necesidades específicas.

- Integración con otras herramientas de Python: se integra bien con otras bibliotecas y herramientas populares de Python, lo que facilita el desarrollo y la implementación de soluciones web completas.

HTML y CSS son lenguajes fundamentales para la construcción de páginas web. Sus motivos y puntos fuertes incluyen:

- Estructura y estilo: HTML se utiliza para definir la estructura de la página web, mientras que CSS se utiliza para controlar el estilo y la apariencia visual. Esto permite crear interfaces de usuario atractivas y funcionales.

- Compatibilidad: HTML y CSS son estándares web ampliamente adoptados, lo que significa que las páginas y aplicaciones creadas con estas tecnologías funcionarán en la mayoría de los navegadores y dispositivos.

- Separación de preocupaciones: La separación de la estructura (HTML) y la presentación (CSS) permite una mayor modularidad y mantenibilidad del código.

JavaScript es un lenguaje de programación ampliamente utilizado en el desarrollo web. Sus motivos y puntos fuertes incluyen:

- Interactividad: JavaScript permite agregar interactividad a las páginas web, realizar validaciones de formularios, manipular elementos de la página y realizar llamadas a APIs, entre otras funcionalidades.

- Amplia adopción: JavaScript es compatible con todos los navegadores modernos y tiene una gran comunidad de desarrolladores, lo que significa que hay abundantes recursos, bibliotecas y frameworks disponibles para su uso.

- Programación del lado del cliente: Con JavaScript, puedes realizar tareas y operaciones en el lado del cliente sin necesidad de comunicarte con el servidor, lo que agiliza la experiencia del usuario.

MetaMask es una extensión de navegador que proporciona una billetera de criptomonedas y una interfaz para interactuar con dApps basadas en Ethereum. Sus motivos y puntos fuertes incluyen:

- Integración con Ethereum: MetaMask facilita la conexión con la red Ethereum y la interacción con contratos inteligentes, lo que te permite desarrollar aplicaciones que interactúen directamente con la blockchain de Ethereum.

- Seguridad y privacidad: MetaMask almacena de forma segura las claves privadas y proporciona una capa adicional de seguridad al permitir que los usuarios revisen y aprueben las transacciones antes de ser enviadas a la red.

- Amplia adopción: MetaMask es una de las carteras de criptomonedas más populares y ampliamente utilizadas en el ecosistema de Ethereum, lo que la convierte en una elección confiable para la interacción con dApps.

MySQL es un sistema de gestión de bases de datos relacional muy utilizado. Sus motivos y puntos fuertes incluyen:

- Confianza y rendimiento: MySQL es conocido por su fiabilidad, escalabilidad y rendimiento en la gestión de grandes volúmenes de datos.

- Lenguaje SQL: MySQL utiliza el lenguaje SQL estándar para interactuar con la base de datos, lo que facilita el almacenamiento y recuperación de datos estructurados.

- Compatibilidad: MySQL es compatible con una amplia gama de lenguajes de programación y frameworks, lo que lo hace fácilmente integrable con aplicaciones desarrolladas en diversos entornos.

5.3 Comunicación entre tecnologías y dependencias

El flujo de comunicaciones de las distintas piezas técnicas queda mejor reflejado en este diagrama específico.

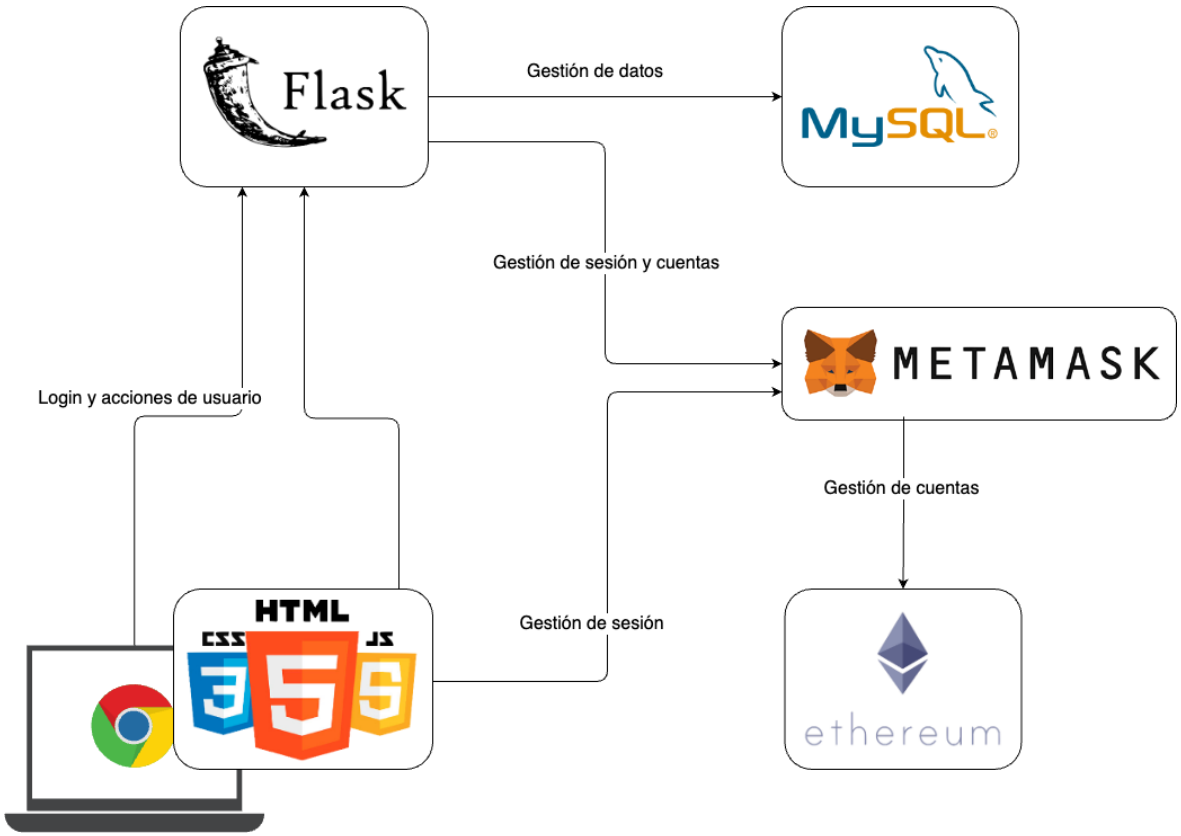


Figura 5. Detalle arquitectura del software

5.3.1 Frontend

El mayor cambio viene en la parte de tecnologías de interfaz, en la cual, también se representa el navegador. Esto nos permite diferenciar las acciones que el usuario realiza en la aplicación (como el login, la gestión de permisos, los cargos… etc.) de las acciones en segundo plano que realiza javascript para controlar que la sesión con metamask sigue activa.

En la parte de implementación (capítulo 7) se detalla la gestión de las sesiones en combinación con flask para evitar accesos no autorizados.

5.3.2 Backend

Esta pieza es la clave de la aplicación, interconecta el resto de los componentes y contiene la mayor parte de la lógica de funcionamiento de la aplicación.

Este backend creado en flask está estructurado lógicamente siguiendo la directriz MVC (Modelo Vista Controlador) la cual nos permite separar la interfaz (agrupada en ficheros estáticos, HTML plantillado con Jinja y hojas de estilo CSS), de los modelos de los objetos albergados en la base de datos y la lógica de consultas a esta.

Se utiliza SQLAlchemy para, aprovechando los modelos, poder operar con varios motores de base de datos sin alterar el código fuente. Las conexiones se realizan contra un servidor de MySQL (en producción) mientras que en el entorno de desarrollo y pruebas se utiliza SQLite por comodidad y agilidad.

El backend también opera contra metamask y Ethereum por dos motivos principales, el primero, necesita verificar la información que le manda el “frontend” sobre el usuario actual y la validez de la sesión, mejorando la seguridad; segundo, necesita listar el conjunto de usuarios activos y las transacciones para poder mostrarla en la interfaz.

6 Entorno de desarrollo

Para desarrollar esta solución, se han necesitado las tecnologías previamente descritas sumando otras complementarias o sustituyendo alguna que permitiera mayor agilidad en el entorno local.

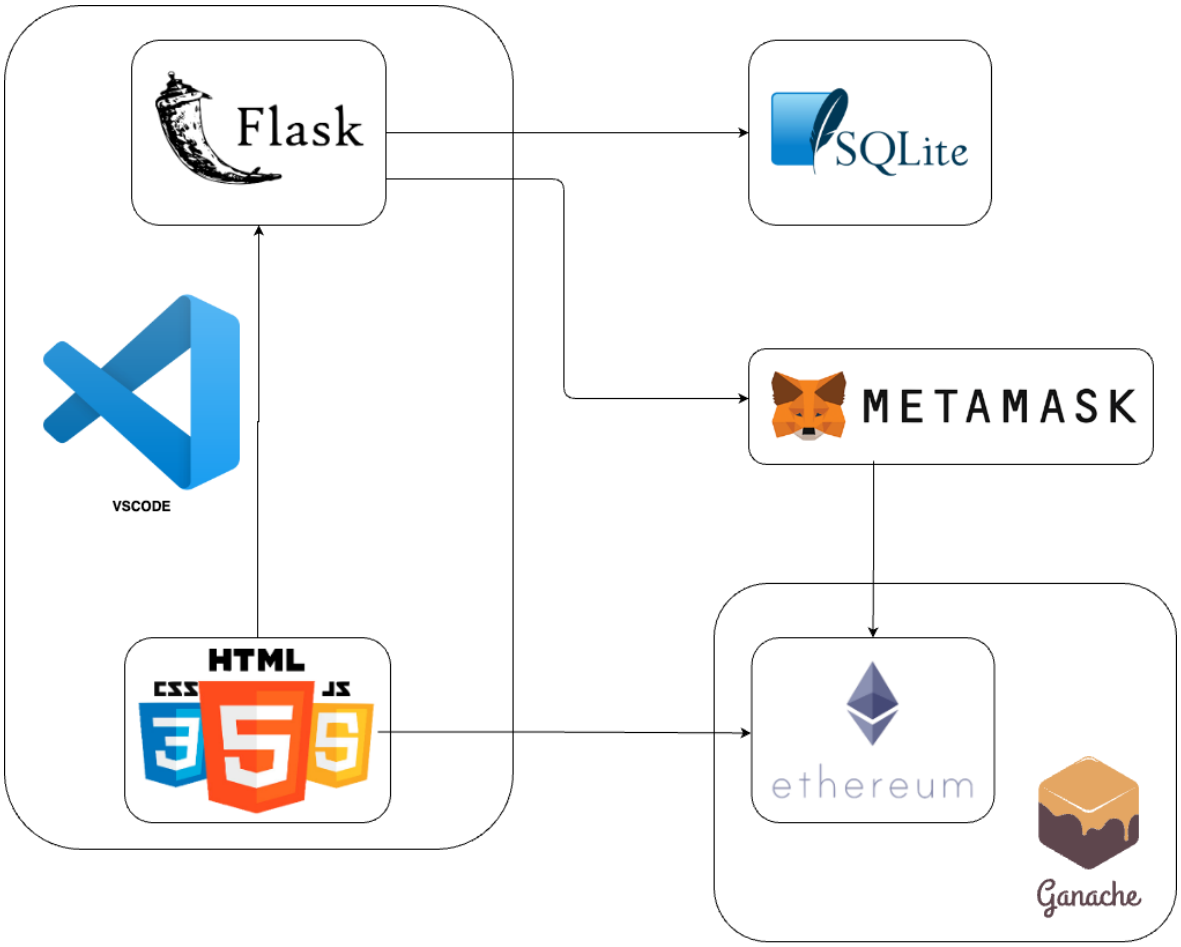


Figura 6. Detalle del entorno de desarrollo

A continuación, se describen individualmente las tecnologías que necesitan un complemento o herramienta específica para facilitar su desarrollo en un entorno local:

6.1 VSCode

Visual Studio Code es el IDE más popular de la actualidad y ampliamente utilizados en la actualidad.

Visual Studio Code es una aplicación de software que proporciona una amplia gama de características y funcionalidades diseñadas para mejorar la experiencia de desarrollo de software. Está disponible de forma gratuita y es compatible con los sistemas operativos Windows, macOS y Linux.

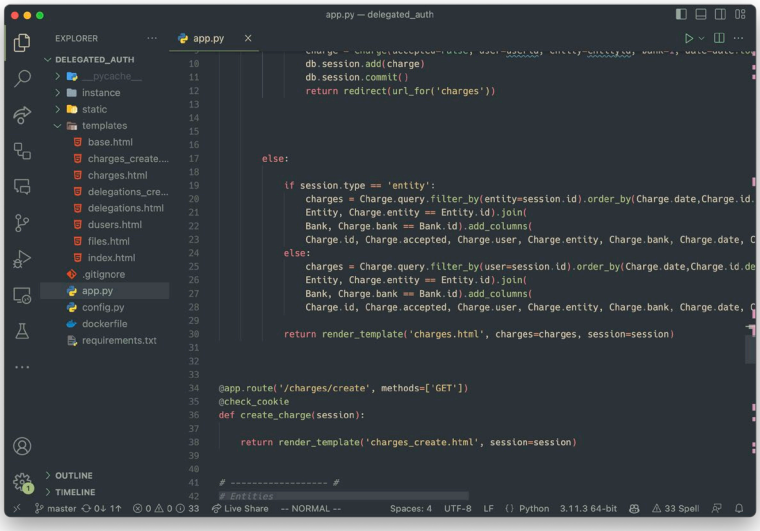


Figura 7. Captura de VSCode

6.2 SQLite

SQLite es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto y autocontenido.

A diferencia de otros sistemas de gestión de bases de datos, SQLite no sigue un modelo cliente-servidor tradicional, sino que es una biblioteca de enlace directo que se incorpora directamente en las aplicaciones. Esto significa que no se necesita un servidor externo para administrar la base de datos, lo cual es perfecto para desarrollar nuestra solución en local.

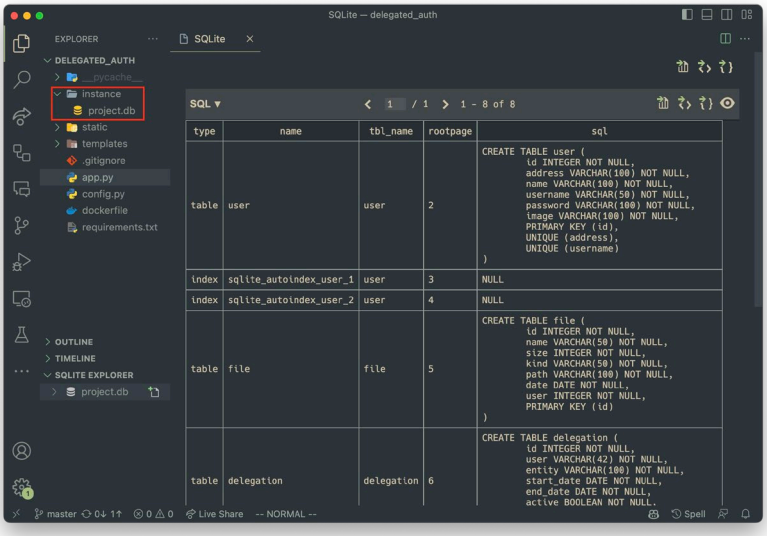


Figura 8. Uso de SQL en VSCode

A efectos prácticos, es un fichero con extensión ".db" en el cual se aloja toda la base de datos. Esto permite trabajar cómodamente en varios equipos en local sin tener un servidor tipo MySQL expuesto a internet. Se puede gestionar con git, aunque su formato de texto no es legible.

En este caso, se ha utilizado la extensión "SQLite" de VSCode para realizar consultas a este fichero como se puede ver en la figura anterior.

6.3 Ganache

Ganache es una herramienta de desarrollo blockchain que se utiliza principalmente para facilitar el proceso de desarrollo, prueba y despliegue de aplicaciones descentralizadas (dApps) basadas en la tecnología de blockchain.

Específicamente, Ganache se enfoca en la creación de un entorno local blockchain que emula una red Ethereum.

Es esencial en cualquier proceso de desarrollo, de lo contrario, cada iteración que tuviéramos con la red de Ethereum implicaría un coste, el cual, dispararía la viabilidad de muchos proyectos.

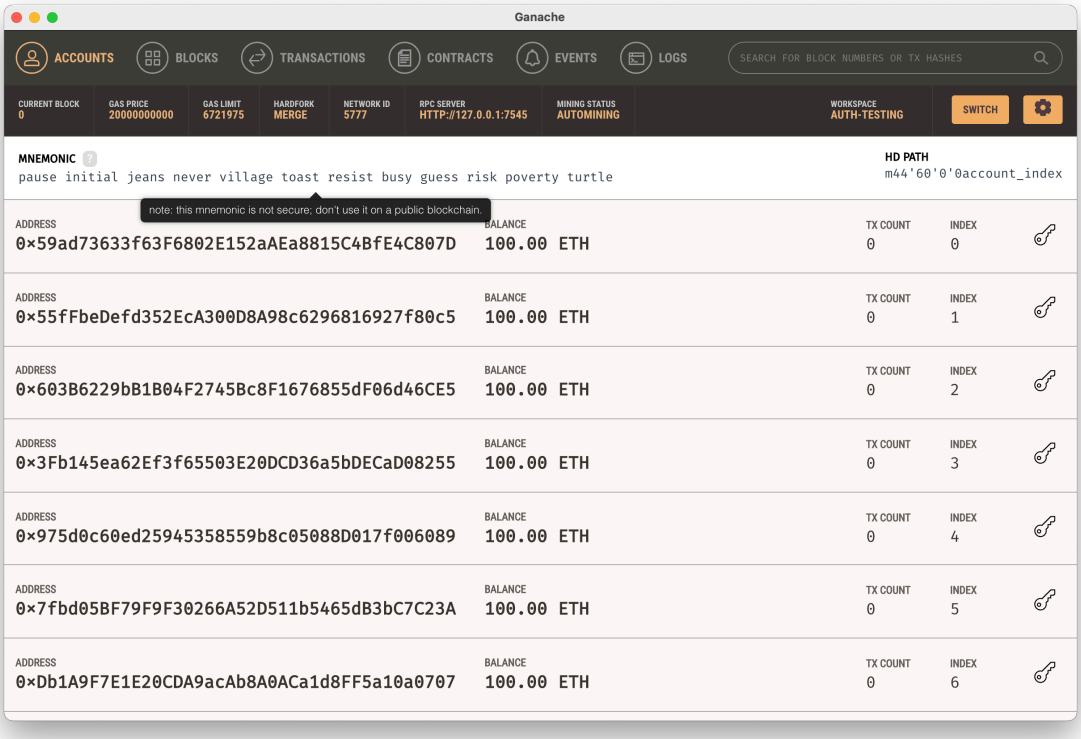


Figura 9. Captura de la aplicación de ganache

Para este entorno se ha creado una red con diversas cuentas para emular la interacción de múltiples usuarios al autenticarse con metamask.

7 Implementación

Para la implementación, se detalla, principalmente en el "frontend" y "backend" los diferentes elementos desarrollados reflejados en la siguiente figura:

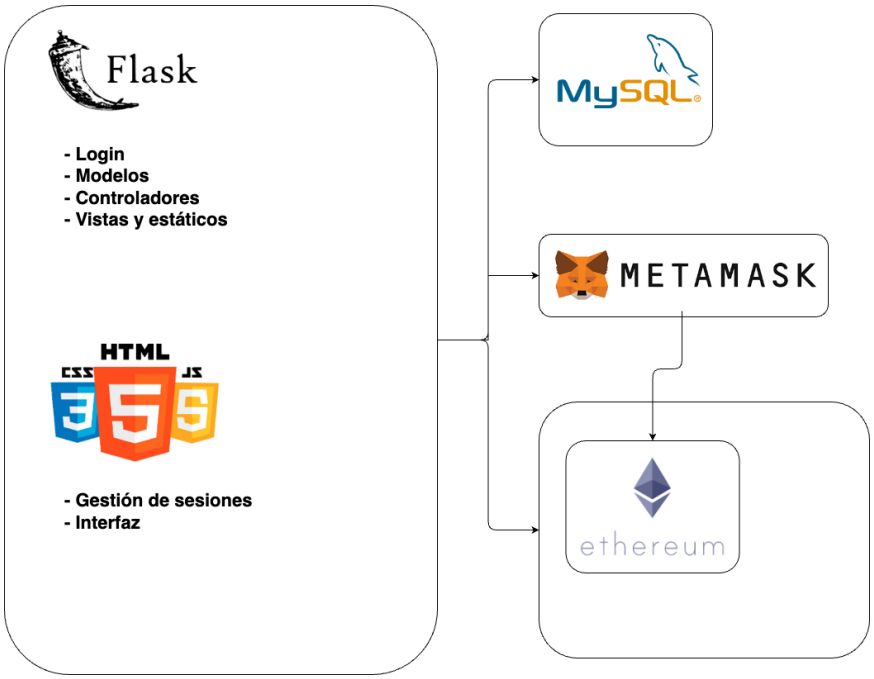


Figura 10. Detalle lógico de la arquitectura de software

El detalle de la implementación de todas las partes se detalla individualmente en los siguientes puntos.

7.1 Creación de una red de Ethereum con Ganache

Como se detalla en el apartado de "Entorno de desarrollo", el proyecto necesita una red de Ethereum local donde poder realizar las pruebas sin imputar ningún coste.

Para instalar ganache podemos acceder a su página oficial y descargarlo para nuestro sistema operativo:

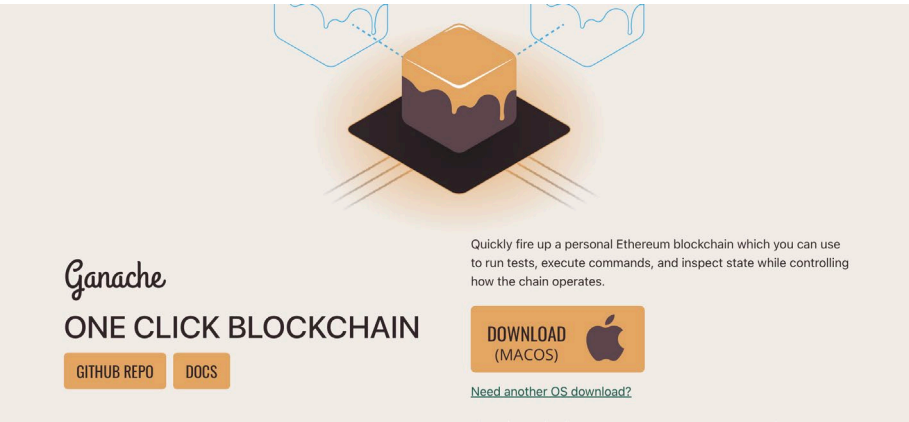


Figura 11. Web de descarga de ganache

Esta herramienta es software libre y podemos encontrar su código fuente en la su página oficial.

Una vez instalada siguiente los pasos del asistente, la aplicación da la opción de crear un espacio de trabajo o retomar una existente.

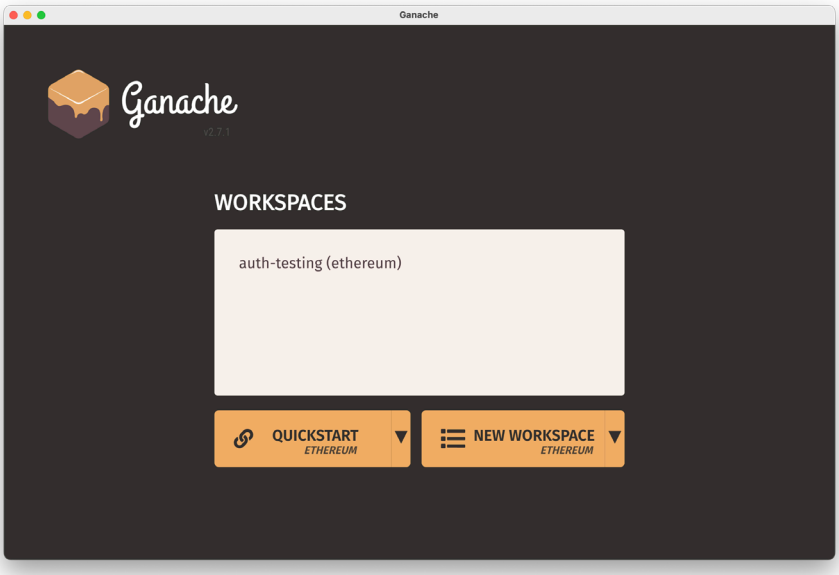


Figura 12. Interfaz de la aplicación ganache

La primera vez, deberemos pulsar “New workspace”. Con cambiar el nombre y pulsar “Start” es suficiente para tener una red de Ethereum operativa, aunque también podemos ajustar el número de cuentas, las claves, configuración del servidor… etc.

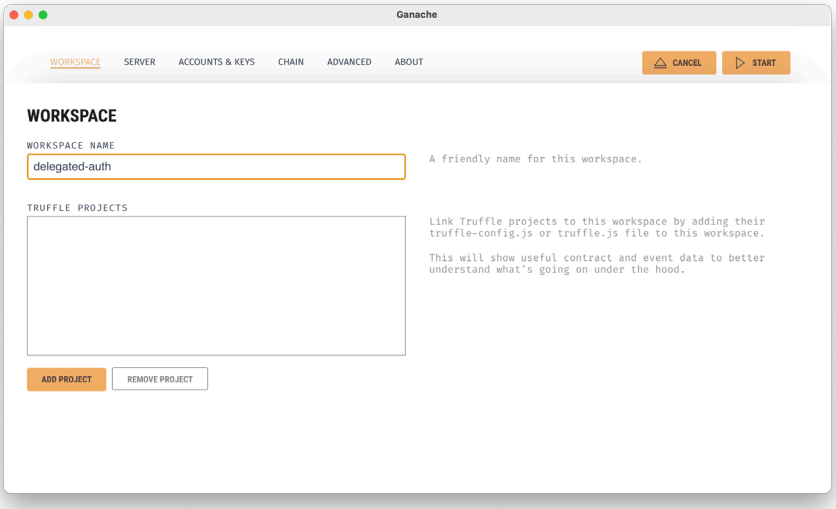


Figura 13. Creación de un workspace en ganache

7.2 Conexión de metamask a la red Ethereum

Con la red de Ethereum operativa, se instala metamask, la cual, usara esta red para validar la autenticación de usuarios.

Esta aplicación funciona como una extensión de navegador compatible con Firefox y los derivados de Chromium (Chrome, Edge… etc.).

Podemos proceder a su instalación a través de su página de descargas, esta detecta automáticamente tu navegador y te instala la versión correspondiente.

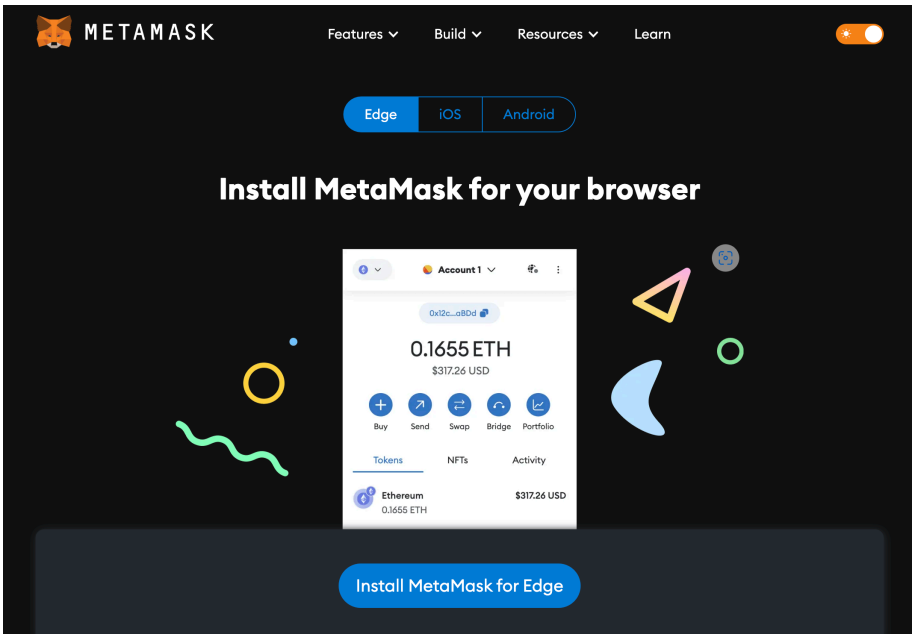


Figura 14. Instalación de metamask en el navegador

Una vez instalada se nos abrirá una ventana con un portal que nos permitirá, o bien crear una cartera o importar una ya existente. Si creamos una nueva, es muy importante recordar la contraseña y las palabras de seguridad, de lo contrario, no podremos recuperar la cuenta.

Si ya tuviéramos una creada, podemos utilizar las palabras de seguridad para importarla y seguidamente nos pedirá una nueva contraseña.

Figura 15. Inicio de sesión en la cartera de metamask

Una vez que hemos creado o importado una cartera, podemos ver la ventana principal de la aplicación. En la siguiente figura, se muestra que, por defecto, esta se conecta a la red de Ethereum. Como se ha explicado en la sección “Entorno de desarrollo” durante todas las pruebas trabajaremos contra la red de Ethereum creada en ganache en el punto anterior.

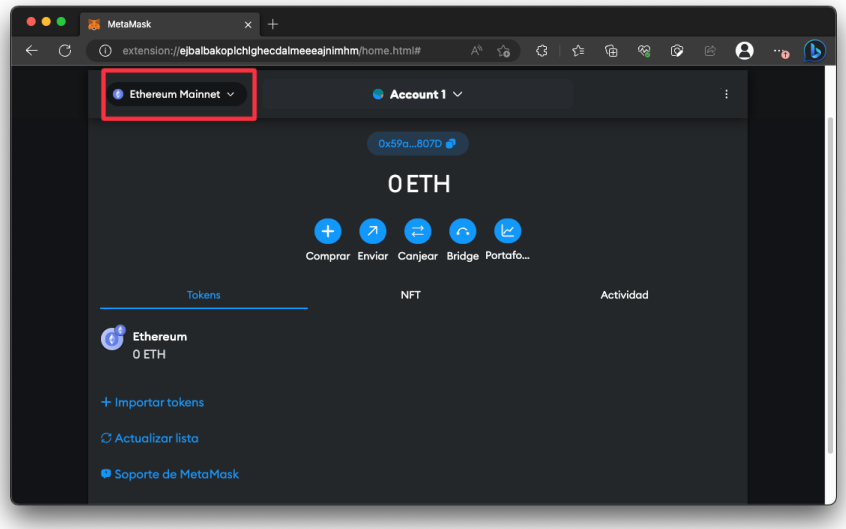


Figura 16. Configuración de metamask con el entorno de pruebas

Lo primero que tendremos que hacer es configurar metamask para que se conecte a nuestra red local de Ethereum. Esto se realiza pulsando en el desplegable de redes y posteriormente en agregar red.

Después de seleccionar la opción "Agregar una red manualmente", nos encontramos con una serie de campos que debemos completar. Estos campos definen la conexión con nuestra red local de Ethereum generada con Ganache. Es crucial configurar estos parámetros correctamente para asegurar la interoperabilidad entre Metamask y nuestra red local:

- **Nombre de la red:** Es recomendable colocar un nombre identificable, como "Ganache Local" o "Ethereum Desarrollo Local".
- **URL RPC:** Esta es la dirección de la interfaz de procedimiento remoto (RPC) de la red. Si estás usando Ganache en tu máquina local, generalmente será "http://127.0.0.1:7545", pero este puerto puede variar dependiendo de tu configuración.
- **ID de Cadena:** Ganache suele utilizar la ID de cadena "1337" para su red local. Sin embargo, es esencial verificar esto en Ganache para asegurarse de que coincida con lo que Metamask espera.
- **Símbolo (opcional):** ETH, dado que estamos trabajando con una red basada en Ethereum.
- **URL del explorador de bloques (opcional):** Puede dejarse en blanco ya que nuestra red local de Ganache no tiene un explorador de bloques en línea.

Una vez que hayamos llenado estos campos, hacemos clic en "Guardar". Metamask debería conectarse automáticamente a nuestra red local de Ethereum, y podremos ver nuestro saldo de Ether (generalmente asignado por Ganache para pruebas) reflejado en la extensión.

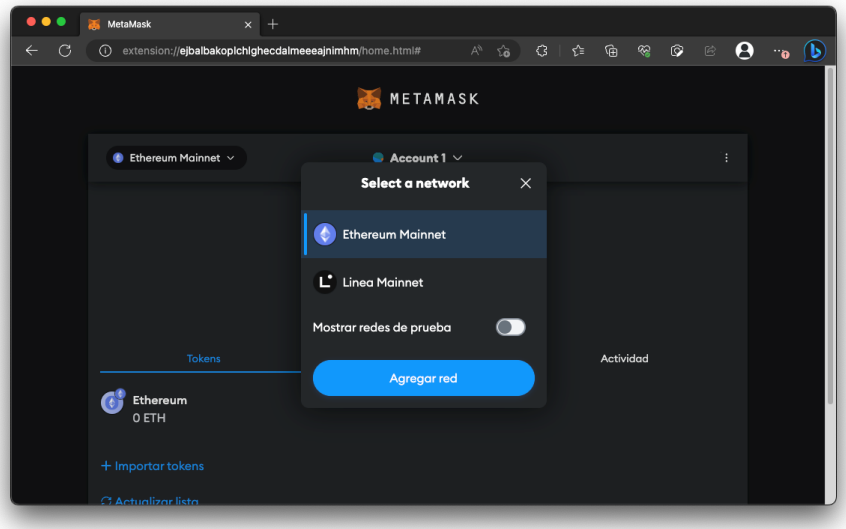


Figura 17. Selección de red local en metamask

En el siguiente listado que nos aparece, nos deslizamos hasta el final y pulsamos en “Agregar una red manualmente”. Aquí podremos configurarla con los siguientes parámetros:

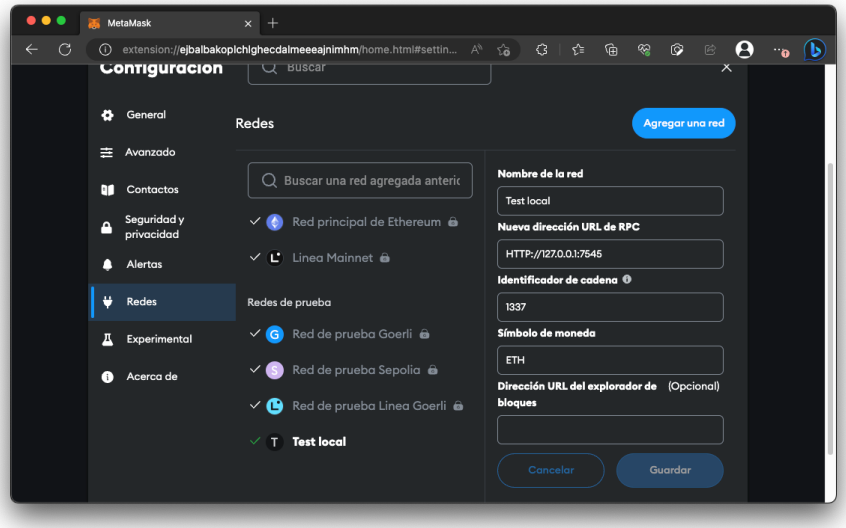


Figura 18. Configuración de red local en metamask

Ahora, toca configurar una o varias cuentas. Teniendo en cuenta que la red de ganache tiene que estar en marcha para que funcione. Podemos pulsar sobre el desplegable de cuentas y pulsar en “importar cuenta”.

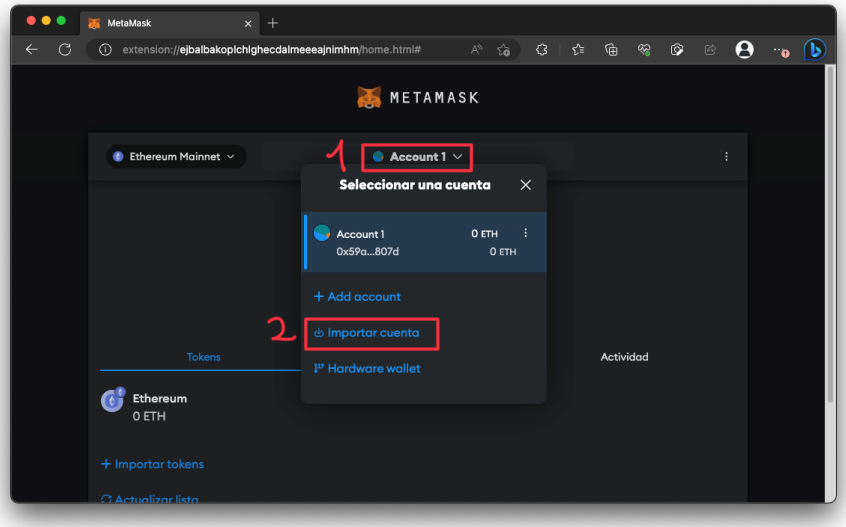


Figura 19. Importar cuentas en metamask

En el siguiente paso nos pide la clave privada de la cuenta, esta se puede obtener en la aplicación de ganache, dentro de la sección de cuentas (accounts) pulsando en el icono de la llave.

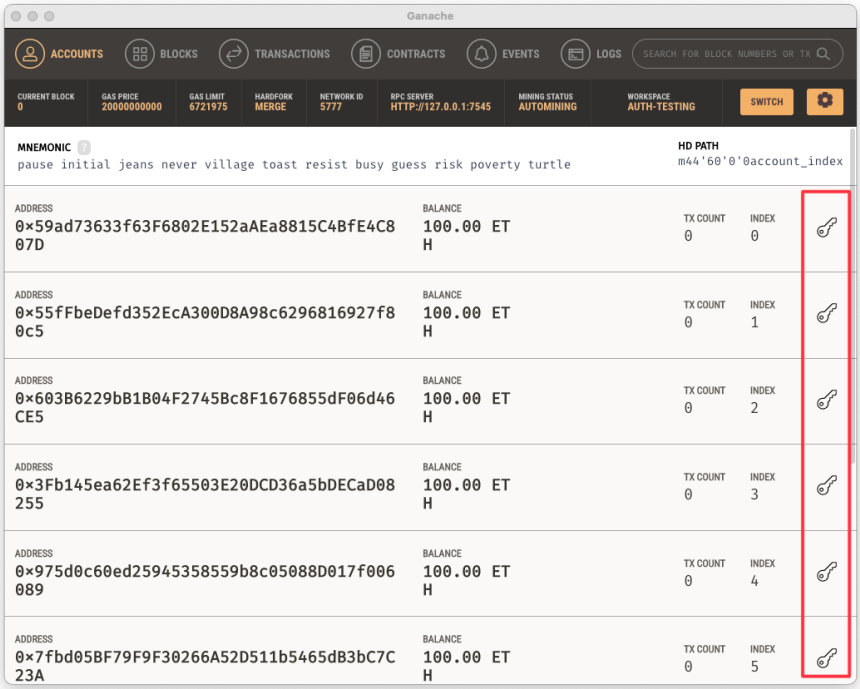


Figura 20. Búsqueda de claves privadas en metamask

Finalmente, copiamos la clave privada de una cuenta de ganache en metamask y pulsamos en importar.

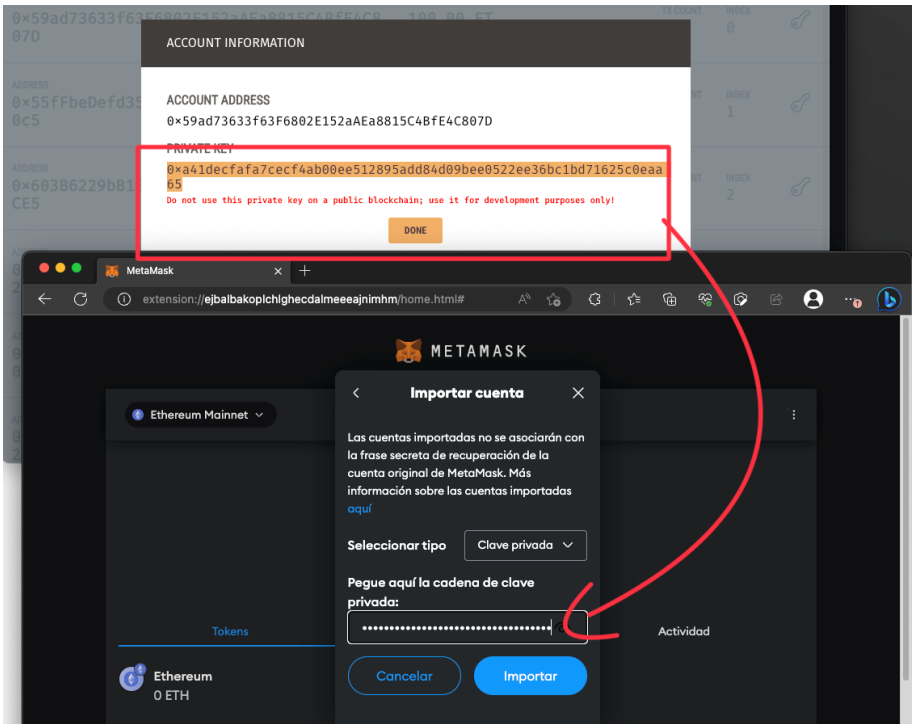


Figura 21. Añadir clave privada a metamask

Una vez importada, ya podríamos verla listada en metamask con los 100ETH que ganache asigna por defecto:

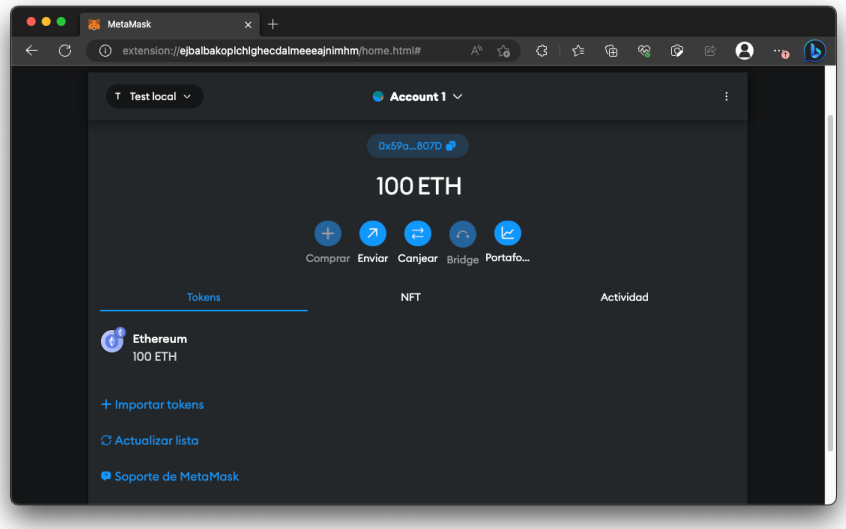


Figura 22. Página de cuentas en metamask

7.3 Gestión de autenticación y sesiones con javascript

Una vez que tenemos configurada nuestra red de Ethereum y la herramienta de metamask, se inicia el desarrollo de la aplicación. Lo primero que tenemos que desarrollar es el sistema de autenticación de la aplicación, a nivel de backend para garantizar la seguridad y en el frontend, para asegurar la usabilidad.

Desde el punto de vista del frontend, se desarrolla un sistema de gestión de sesiones con javascript y una ventana de login para que el usuario interactúe con la aplicación.

Para el panel del login se crea una vista simple con un único botón de "login" que lanzará eventos contra la extensión de metamask para validar que el usuario esta autenticado en la red de Ethereum.

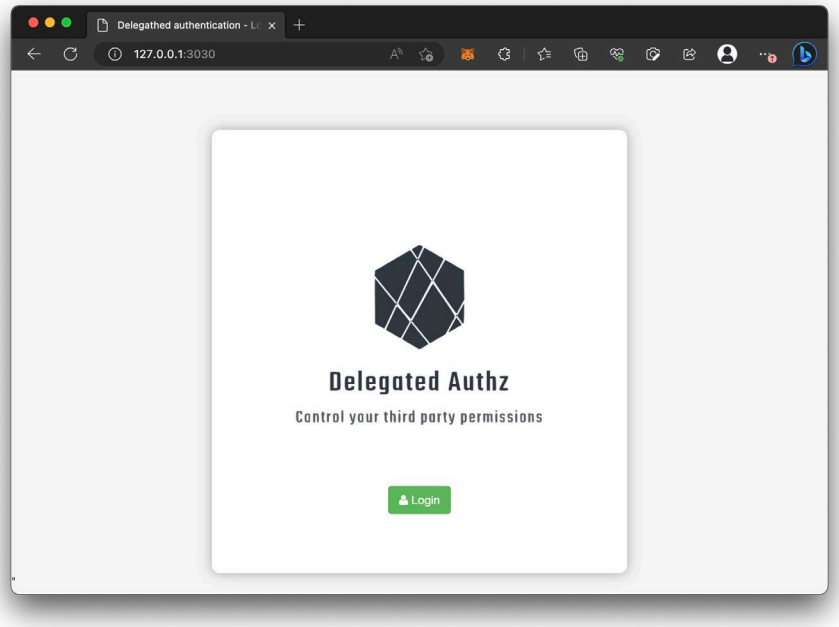


Figura 23. Página de inicio de la aplicación

Al pulsar el botón de login, se llama mediante javascript a la extensión de metamask para verificar el usuario esta autenticado, el backend hará un doble check verificando que además de tener una cuenta en la red de Ethereum también está autorizado a entrar y que tipo de usuario es.
En la siguiente figura se muestra un fragmento del código fuente del fichero "index.html", el cual, ejerce de portal de login en la aplicación y es lo primero que visualiza el usuario.

index.html — delegated_auth

index.html templates

```
    <title>Delegathed authentication - Login</title>
    </title>
    <link href="https://netdna.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/login.css') }}">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js/dist/web3.min.js"></script>


</head>

<body>
    <link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css" rel="stylesheet">
    <div class="container">
        <div class="login-wrapper center-block">

            <div class="logo text-center">
                <img class="img-profile img-circle img-responsive center-block"
                    src="img/logo.png" alt>

            </div>

            <div class="content-panel text-center">
                <div class="content-header-wrapper">

                    <div class="actions">
                        <button id="login-button" class="btn btn-success login-button"><i class="fa fa-user"></i> Login</button>
                    </div>
                </div>

            </div>

        </div>

    </div>
    <script src="https://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="https://netdna.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <script type="text/javascript">
        $(function () {
            $("[data-toggle='tooltip']").tooltip();
        })
    </script>
    <script  src="{{ url_for('static', filename='js/login.js') }}"></script>"
</body>

</html>
```

Figura 24. Código de la página inicial de la aplicación

En las últimas líneas del fichero “index.html” se importa la librería “web3” para javascript (estándar en la comunicación con sistemas web3 y metamask en este caso) y, también, el script de gestión de sesiones desarrollado para este proyecto.

Este script, llamado “login.js” localizado en el proyecto en “static/js/login.js”. Este se encarga de lanzar la lógica de la gestión de sesión cuando se pulsa en el botón login.

login.js — delegated_auth

login.js static/js

```
document.getElementById('login-button').addEventListener('click', async () => {

    if (window.ethereum) {
        window.web3 = new Web3(ethereum);
        try {
            // Solicitar acceso a la cuenta si es necesario
            await ethereum.enable();

            // Cuenta actual
            const accounts = await ethereum.request({ method: 'eth_requestAccounts' });

            // Redirigir a la página de landing

            // Crear cookie con la cuenta
            document.cookie = "account=" + accounts[0] + "; path=/";

            window.location.href = "files";

        } catch (error) {
            // El usuario denegó el acceso a la cuenta
            console.error(error)
            console.error("El usuario denegó el acceso a la cuenta");
        }
    }
    // Si no se inyectó Web3 (navegador antiguo), crea una instancia de Web3 con el proveedor HttpProvider
    else {
        console.log('No se detectó MetaMask. Deberías considerar probar MetaMask!');
        window.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
    }


    const accounts = await ethereum.request({ method: 'eth_requestAccounts' });

});
```

Figura 25. Lógica del script de login

La lógica de gestión de sesiones valida, usando la librería web3 comentada anteriormente, que existe algún aplicativo de tipo Ethereum (llamada que invoca el navegador y que extensiones compatibles con el protocolo como metamask capturan). Si existe una cartera de Ethereum con la sesión iniciada en metamask nos permitirá entrar en la aplicación, por el contrario, nos devolverá a la página del login.

Por último, escribe el nombre de la cuenta en una cookie para facilitar la experiencia de usuario y guardar la sesión activa hasta que el usuario la cierre manualmente desde las opciones de la aplicación. Todas estas funcionalidades se ven en detalle de la sección de casos de uso desde una perspectiva de usuario.

7.4 Creación y gestión de la autenticación en flask

El backend debe tener siempre la última palabra en materia de seguridad, de lo contrario, se podrían realizar acciones sin el permiso del usuario. Para ello, se toman dos acciones, primero leer las cookies de usuario que hace una petición para validar que cuenta es y verificar en Ethereum que la cuenta existe y la sesión sigue activa. **Si una de estas dos condiciones no se cumple en cada petición** se redirige al usuario al portal de login.

Para que esto se pudiera aplicar a las diferentes rutas y métodos que el API ofrece para su funcionamiento, se opta por utilizar decoradores de Python.

Un decorador en Python es una función especial que se utiliza para modificar o extender el comportamiento de otras funciones o métodos. Esto se logra mediante el uso de funciones anidadas y retornando funciones.

Para este caso, nos permitiría, en cualquiera de las rutas del api, procesar las cookies que el usuario introduzca y aplicar las validaciones de seguridad sin tener que duplicar código ni hacer más complejos los métodos existentes.

app.py — delegated_auth

app.py

```python
# ------------------ #
# SECURITY FUNCTIONS
def check_address(address):
    session = User.query.filter_by(address=address).first()

    if session == None:
        session = Entity.query.filter_by(address=address).first()

        if session != None:
            session.type = 'entity'
        else:
            session = False
    else:
        session.type = 'user'

    return session


# Decorator function to check cookies
def check_cookie(f):
    @wraps(f)
    def contenedor(*args, **kwargs):
        address = request.cookies.get('account')
        session = check_address(address)

        if not session:
            return render_template('index.html')

        return f(session=session, *args, **kwargs)
    return contenedor
```

Figura 26. Funciones de seguridad en la aplicación

La función “check_cookie” acepta por parámetros otra función y, utilizando las variables reservadas de Python *args y **kwargs podemos coger los parámetros de la función que se recibe (es decir, cookies, parámetros que el usuario introduce, etc.…).

Tras verificar que existe una cookie con una cuenta válida, apoyándose en la función check_address, devuelve la sesión y los parámetros a la función recibida por parámetros, actuando como una barrera de seguridad antes de que los métodos decorados se ejecuten.

Finalmente, podemos hacer que esta función se aplique a cualquier método “decorándolos”. En Python se puede realizar invocando a la función con una arroba, por ejemplo:

Figura 27. Decorador que comprueba las cookies en flask

Esto ejecutará toda la lógica comentada antes de que se ejecute esta ruta de flask, la cual, lista todos los usuarios de la aplicación. Esta capacidad es una de las más potentes de Python y permite mantener el código muy limpio.

7.5 SQLAlchemy y modelos de base de datos

Antes de gestionar toda la lógica de usuarios, delegaciones, cobros… etc., debemos de crear una base de datos relacional que soporte toda la operativa. Para facilitar el proceso de usa SQLAlchemy.

Esta tecnología es un sistema ORM, el cual, simplemente definiendo los modelos de los datos que queremos guardar nos genera el esquema de la base de datos y nos permite ejecutar consultar independientemente del motor de base de datos.
Esta última parte es especialmente importante ya que permite trabajar en fase de desarrollo con tecnologías como SQLite y, una vez que queremos hacer productiva la aplicación, usar más potentes como SQLServer, PostgreDB o MySQL.

7.5.1 Enumeración de los modelos y tipos de objetos

El funcionamiento de la aplicación se sostiene en los siguientes objetos de información:

- User o usuario. Contiene la información esencial de un usuario y lo relaciona con su cartera de Ethereum.

- File o fichero. Contiene un archivo variopinto sobre un cargo, como un recibo o factura.
- Delegation o delegación. Almacena el permiso que da un usuario a una entidad para automatizar un cargo.
- Entity o entidad. Alberga los detalles de una entidad (por ejemplo, la agencia tributaria) y la asocia una cartera de Ethereum.
- Bank o banco. Contiene los detalles de uno de los múltiples bancos que puede configurar un usuario para recibir los cargos.
- Charge o cargo. Almacena la información de un cargo realizado de una entidad a un usuario.

7.5.2 Usuario

El modelo del usuario actúa como una representación de un individuo dentro de la plataforma. Guarda información básica sobre el usuario, su dirección de cartera Ethereum y otros detalles pertinentes.

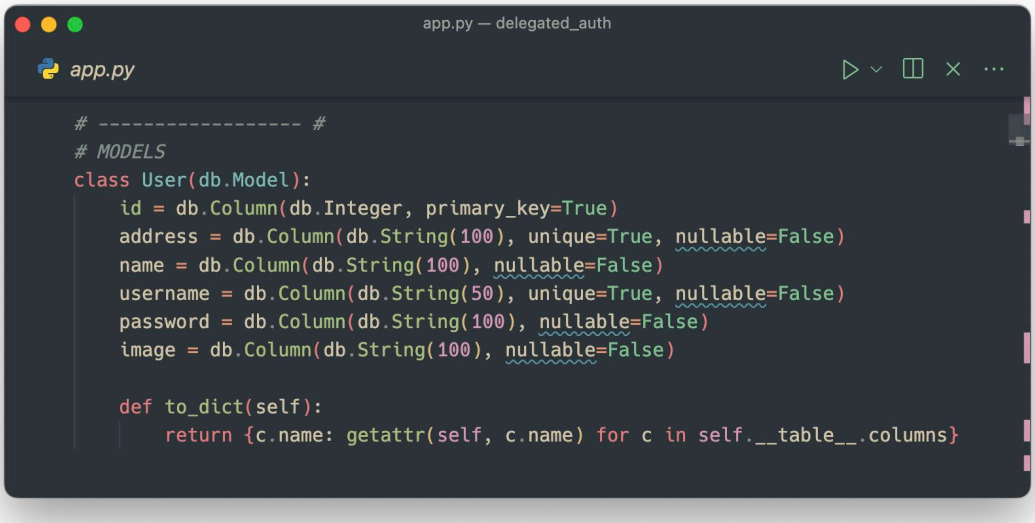


Figura 28. Modelo del dato usuario

7.5.3 Fichero

El modelo "Fichero" representa un archivo almacenado por un usuario en la plataforma. La relación con el modelo "User" asegura que cada archivo esté asociado con un usuario específico, garantizando la privacidad y segregación.

La clase fichero contiene el nombre, la ruta, tamaño y el usuario al que está asociado dicho fichero, permitiendo una segregación de permisos y evitando que un usuario pueda ver ficheros que no le pertenecen. Por último, alberga la ruta del servidor donde se aloja el fichero y la fecha de su subida.

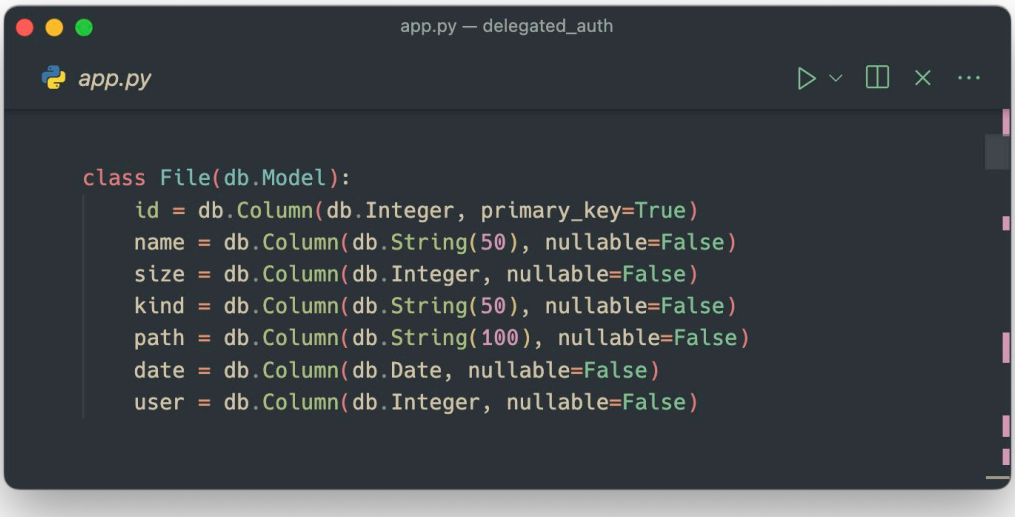


Figura 29. Modelo del dato fichero

7.5.4 Delegación

Permite contener los detalles de una delegación de permisos de un usuario a una entidad. Contiene los campos de ambas partes para relacionarlos individualmente, así como la expiración del permiso, su inicio y un campo lógico para activar o desactivar el permiso manualmente a voluntad del usuario.

app.py — delegated_auth

app.py

```python
class Delegation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user = db.Column(db.String(42), nullable=False)
    entity = db.Column(db.String(100), nullable=False)
    start_date = db.Column(db.Date, nullable=False)
    end_date = db.Column(db.Date, nullable=False)
    active = db.Column(db.Boolean, nullable=False)
```

Figura 30. Modelo del dato delegación

7.5.5 Entidad

Relaciona entidades públicas, como por ejemplo agencia tributaria, SEPE u otras, con una cartera de Ethereum. Esto permite, además de controlar la lógica de delegaciones, albergar una imagen y nombre para mejorar la estética de la aplicación.

El modelo de entidad está diseñado para contener información relevante sobre las diferentes entidades que pueden emitir cargos hacia los usuarios. Al igual que el modelo de usuario, contiene detalles como el nombre de la entidad, dirección, número de teléfono y una cartera asociada de Ethereum que permitiría identificar y realizar transacciones de dicha entidad en la red de Ethereum.

app.py — delegated_auth

app.py

```python
class Entity(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(42), nullable=False)
    image = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(100), nullable=False)

    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Figura 31. Modelo del dato entidad

Esto asegura que cada entidad tenga una cartera única asociada y permite la verificación y autenticación de transacciones originadas desde estas entidades.

7.5.6 Banco

Todos los cargos que se realizan en la aplicación necesitan un banco autorizado donde realizar los cargos. Este modelo se encarga que relacionar un banco con un usuario, así como activarlo y definir su prioridad (esto hará que un cargo se realice en un banco u otro, a preferencia del usuario). Por último, contiene la imagen y nombre que, al igual que en las entidades, mejora la apariencia de la aplicación.

app.py — delegated_auth

app.py

```python
class Bank(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    user = db.Column(db.Integer, nullable=False)
    image = db.Column(db.String(100), nullable=False)
    active = db.Column(db.Boolean, nullable=False)
    priority = db.Column(db.Integer, nullable=False)
```

Figura 32. Modelo del dato banco

7.5.7 Cargos

Los cargos guardan la relación ternaria del cobro de una entidad a un usuario y el banco en el que se hizo efectiva, además del importe, descripción, fecha y si se realizó correctamente o no.

El modelo de cargos es esencialmente el registro de una transacción individual realizada por una entidad a un usuario. Contiene campos como la cantidad, la fecha de la transacción, la entidad emisora y el usuario receptor, así como cualquier otra información relevante relacionada con esa transacción en particular, como detalles del producto o servicio, número de referencia, entre otros.

app.py — delegated_auth

app.py

```python
class Charge(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    accepted = db.Column(db.Boolean, nullable=False)
    user = db.Column(db.String(42), nullable=False)
    entity = db.Column(db.String(100), nullable=False)
    bank = db.Column(db.String(100), nullable=False)
    date = db.Column(db.Date, nullable=False)
    amount = db.Column(db.Integer, nullable=False)
    description = db.Column(db.String(100), nullable=False)
```

Figura 33. Modelo del dato cargo

Este modelo permite realizar un seguimiento de todas las transacciones y cargos emitidos a los usuarios y es fundamental para la lógica de negocio de la aplicación.

Estos modelos proporcionan una base sólida para la construcción de la lógica de negocio en la aplicación. Las relaciones entre ellos, como se refleja en los campos **db.ForeignKey** y **db.relationship**, permiten una fácil navegación y consulta entre los datos relacionados, haciendo que las operaciones CRUD y las consultas más avanzadas sean más simples y eficientes.

7.6 Creación de base de datos de prueba automática

Durante las pruebas de desarrollo, tener unos datos de prueba es vital para no perder mucho tiempo y, si en algún momento hay que borrar las pruebas, que se genere automáticamente se vuelve imprescindible.

Para ello, se utiliza una función especial en flask que nos permite ejecutar una función antes de que todo el servicio se ponga en marcha. En ella, se comprueba si la base de datos de SQLite alojada localmente en “instance/project.db” existe y, sino es el caso, la crea automáticamente.

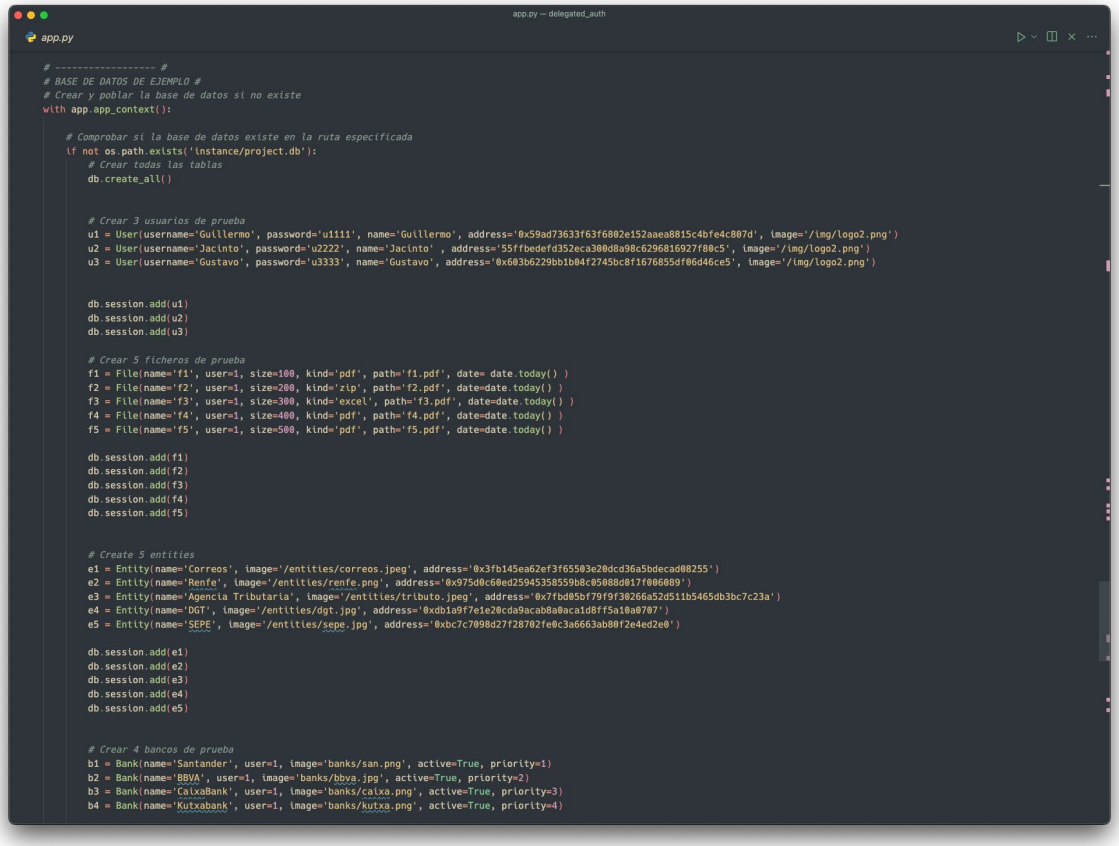


Figura 34. Carga de base de datos automática

Para poblar la base de datos, con el sistema ORM solo hay que crear objetos con los modelos de los distintos tipos de datos y hacer un "commit" con lo cambios. Por debajo, SQLAlchemy gestiona todo para crearnos automáticamente las correspondientes tablas y rellenarlas con los datos pasados en el ejemplo anterior.

7.7 Consultas a la base de datos

Toda la lógica de la aplicación se ha construido en las diferentes rutas del API. Cada una de estas rutas es una función, la cual, puede responder a más de un método HTTP (Get o Post).

Estas se pueden definir en flask con el decorador @app.route() como en el siguiente ejemplo:

```
# ------------------ #
# ROUTES
@app.route('/', methods=['GET'])
def dashboard():
    return render_template('index.html')


@app.route('/files', methods=['GET', 'POST'])
@check_cookie
def files(session):
    if request.method == 'POST':
        pass
    else:
        files = File.query.filter_by(user=session.id).all()
        return render_template('files.html', files=files, session=session)
```

Figura 35. Rutas de consulta a la base de datos

En los parámetros introducimos la URL con la que se va a invocar la función y los métodos HTTP soportados.
Los controladores más importantes, los cuales contienen la lógica de la aplicación, se detallan en los siguientes puntos.

Para ejemplificar esta parte y detallar el sistema de funcionamiento de SQLAlchemy, se va a utilizar la ruta "delegations", una de las muchas que se han utilizado para el funcionamiento total de la aplicación.
"Delegations" nos permite listar las delegaciones del usuario actual usando el método GET y crear nuevas usando el método POST.

```python
# ------------------ #
# DELEGATIONS
@app.route('/delegations', methods=['GET', 'POST'])
@check_cookie
def delegations(session):

    if request.method == 'POST':

        entity = request.form.get('entity')
        r_end_date = request.form.get('end_date')

        end_date = date(int(r_end_date[0:4]), int(r_end_date[5:7]), int(r_end_date[8:10]))

        delegation = Delegation(
            user=1,
            entity=entity,
            start_date= date.today(),
            end_date=end_date,
            active=True
        )

        db.session.add(delegation)
        db.session.commit()

        return redirect(url_for('delegations'))

    else:

        delegations = Delegation.query.filter_by(user=session.id).join(
            Entity, Delegation.entity == Entity.id).add_columns(
            Delegation.id, Delegation.user, Delegation.entity, Delegation.start_date, Delegation.end_date, Delegation.active, Entity
        banks = Bank.query.order_by(Bank.priority).all()
        return render_template('delegations.html', delegations=delegations, banks=banks, session=session)
```

Figura 36. Función de delegación de permisos en Python

Utilizando las funciones de gestión descritas anteriormente podemos obtener en todas las funciones el usuario actual que está consultando el API. Se puede ver la variable “session” como parámetro de todos los controladores que se van a mostrar.

Aquí es importante filtrar las delegaciones del usuario actual para asegurar la correcta lógica y seguridad de la información de la aplicación.

También podríamos crear una delegación utilizando los parámetros que nos introduce el usuario. La ventaja de utilizar un sistema ORM es que se encarga por nosotros de la seguridad, aunque, por consiguiente, tenemos que aprender a realizar las consultas de la forma que marca el sistema en vez de usar SQL.

Por ejemplo, para hacer un “Select * FROM table” en SQLAlchemy solo tenemos que coger el modelo de los datos que queremos obtener e invocar la siguiente función:

```python
banks = Bank.query.order_by(Bank.priority).all()
```

Figura 37. Ejemplo de select en SQLAlchemy

También podríamos utilizar modificadores como “order_by”, ejemplificado en la consulta anterior.

La parte más compleja puede darse al realizar consultas que involucren a varias tablas. Por ejemplo, la parte de delegaciones tiene que garantizar que solo sea del usuario actual y realizar un join con la tabla "Entity", la cual, contiene la información de la entidad a la que se está delegando el permiso. Puede ser en estas situaciones donde se vuelva un poco más complejo que una SQL tradicional al cambiar ligeramente el sistema de funcionamiento:

```
delegations = Delegation.query.filter_by(user=session.id).join(
    Entity, Delegation.entity == Entity.id).add_columns(
    Delegation.id, Delegation.user, Delegation.entity, Delegation.start_date, Delegation.end_date, Delegation.active, Entity.image).all()
```

Figura 38. Ejemplo de query compleja en SQLAlchemy

7.8 Creación de las vistas y estáticos en flask

Toda la renderización de las vistas se realiza utilizando estáticos y plantillas en flask. Es importante diferenciar estas partes:

- Las plantillas en flask permite definir HTLM reutilizable por diferentes vistas, el cual, se puede modificar utilizando variables, condicionales y bucles.

- Los estáticos, son ficheros que no se modifican en tiempo de ejecución. Aquí podemos agrupar el código javascript, las imágenes y los ficheros CSS.

7.8.1 Plantillas

Estas plantillas combinan código HTML y sintaxis de Jinja para generar dinámicamente las distintas interfaces de la aplicación.

Jinja es un motor de plantillas, pensado para trabajar con una sintaxis muy similar a Python. Esta tecnología permite definir variables y condiciones para, cuando toque renderizar, aplicar cierta lógica y modificaciones.

Por ejemplo, para cargar un fichero .html, podemos definir un objeto template y luego renderizando pasando diferentes variables a la función.

To load a template from this environment, call the **`get_template()`** method, which returns the loaded **`Template`**.

```
template = env.get_template("mytemplate.html")
```

To render it with some variables, call the **`render()`** method.

```
print(template.render(the="variables", go="here"))
```

Figura 39. Carga de plantillas en flask.

En este proyecto, se ha usado en combinación con las rutas de flask para devolver al navegador esta r renderización. Por ejemplo, la vista de los usuarios delegados:

```
# ------------------ #
# Delegated users
@app.route('/dusers', methods=['GET'])
@check_cookie
def dusers(session):
    users = User.query.all()

    return render_template('dusers.html', users=users, session=session)
```

Figura 40. Renderizado de estáticos en flask

Todas las interfaces de la aplicación se han construido en torno a una plantilla base. Esta contiene el menú, cabeceras y barra lateral, los cuales, están compuestos de variables que se modifican desde otro fichero html al extender de esta plantilla base.

```
<head>
    <meta charset="utf-8">


    <title>{% block title %}{% endblock %} - Delegated auth</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://netdna.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/landing.css') }}">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js/dist/web3.min.js"></script>

</head>

<body>
    <link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css" rel="stylesheet">
    <div class="container">
        <div class="view-account">
            <section class="module">
                <div class="module-inner">
                    <div class="side-bar">
                        <div class="user-info">…
                        </div>
                        <nav class="side-menu">
                            <ul class="nav">
                                {% if session.type == 'user' %}
                                <li class="{% block permissions %}{% endblock %}"><a href="/delegations"><span class="fa fa-cog"></span> Permisos</a></li>
                                {% endif %}

                                <li class="{% block charges %}{% endblock %}"><a href="/charges"><span class="fa fa-credit-card"></span> Cargos</a></li>
                                <li class="{% block files %}{% endblock %}"><a href="/files"><span class="fa fa-th"></span> Ficheros</a></li>

                                {% if session.type == 'entity' %}
                                <li class="{% block del_users %}{% endblock %}"><a href="/dusers"><span class="fa fa-user"></span> Usuarios</a></li>
                                {% endif %}

                                <li id="log-out"><a href="#"><span class="fa fa-clock-o" ></span> Cerrar sesión</a></li>
                            </ul>
                        </nav>
                    </div>
                    <div class="content-panel">
                        <div class="content-header-wrapper">
                            <h1 class="title">{{ self.title() }}</h2>
                            <div class="actions">
                                <a href="{{ self.action_url() }}">

                                    {% if action %}
                                    <button class="btn btn-success"  >
                                        <i class="fa fa-plus"></i> {{ self.action_name() }}
                                    </button>
                                    {% endif %}
```

Figura 41. Plantilla base de la interfaz html

En la figura anterior, se puede ver la plantilla base, la cual contiene variables para definir el título de la página, mostrar unos elementos del menú u otros en función del tipo de usuario... etc.

Podemos usar las variables de bloque para añadir variables simples o fragmentos de HTML completos. Por ejemplo, para el título se utiliza esto:

```
<title>{% block title %}{% endblock %} - Delegated auth</title>
```

Figura 42. Variables en jinja

Jinja también permite aplicar condicionales, por ejemplo, si el usuario es una entidad o no, se muestra unos elementos en el menú u otros:

```
<nav class="side-menu">
    <ul class="nav">
        {% if session.type == 'user' %}
        <li class="{% block permissions %}{% endblock %}"><a href="/delegations"><span class="fa fa-cog"></span> Permisos</a></li>
        {% endif %}

        <li class="{% block charges %}{% endblock %}"><a href="/charges"><span class="fa fa-credit-card"></span> Cargos</a></li>
        <li class="{% block files %}{% endblock %}"><a href="/files"><span class="fa fa-th"></span> Ficheros</a></li>

        {% if session.type == 'entity' %}
        <li class="{% block del_users %}{% endblock %}"><a href="/dusers"><span class="fa fa-user"></span> Usuarios</a></li>
        {% endif %}

        <li id="log-out"><a href="#"><span class="fa fa-clock-o" ></span> Cerrar sesión</a></li>
    </ul>
</nav>
```

Figura 43. Condicionales en jinja

Por último, también permite recibir una lista de elementos permitiendo crear bucles. Todas las vistas tienen una sección de listado de cuadrícula para mostrar un conjunto de elementos. Por ejemplo, si se recibiera un listado en estos bloques se escribirían tantas veces como número de elementos.

```
<div class="drive-wrapper drive-grid-view">
    <div class="grid-items-wrapper">
        {% block grid_items %}{% endblock %}
    </div>
</div>
{% block list_items %}
{% endblock %}
</div>
```

Figura 44. Bucles en jinja

Ahora bien, la plantilla no contiene el dato final. Cada vista que extienda de ella tendrá que definir todas las variables mencionadas. Estas vistas son:

- Files – Listado de ficheros.
- Dusers – Usuarios delegados (solo para entidades).
- Delegations – Delegaciones. (Solo usuarios)
- Delegations_create – Formulario para crear delegaciones. (Solo usuarios)
- Charges – Cargos o cobros que realiza una entidad a un usuario.
- Charges_create – Formulario que permite crear nuevos cargos (solo entidades).

Todas estas vistas extienden de la plantilla base y configuran las variables pertinentes para rellenar el título, menú y contenido de la página correspondiente. Ejemplo de la vista de ficheros (files.html):

```
{% extends "base.html" %}

{% block title %}
Ficheros
{% endblock %}


{% block action %}
True
{% endblock %}

<!-- ACTIONS -->
{% set action = True  %}


{% block action_name %}
Subir fichero
{% endblock %}

{% block action_url %}
/files/upload
{% endblock %}
```

Figura 45. Ejemplo de invocación de una plantilla con carga de variables en jinja

Podemos ver como extiende de la plantilla base.

Cambia el título de la página

Permite realizar acciones

Cambia el contenido del botón

Modifica la URL a la que apunta el botón.

Además de las variables, también define bloques de código HTML específico de la página de ficheros:

```
{% block grid_items %}

{% for file in files %}

    <div class="drive-item module text-center">
        <div class="drive-item-inner module-inner">
            <div class="drive-item-title"><a href="#">{{ file.name }}</a></div>
            <div class="drive-item-thumb">
                <a href="#"><i class="fa fa-file-{{ file.kind }}-o text-primary"></i></a>
            </div>
        </div>
        <div class="drive-item-footer module-footer">
            <ul class="utilities list-inline">
                <li><a href="#" data-toggle="tooltip" data-placement="top" title
                        data-original-title="Download"><i class="fa fa-download" style="font-size: 20px; color: green;"></i></a>
                </li>
                <li>
                    <a href="#" data-toggle="tooltip" data-placement="top" title
                        data-original-title="Delete"><i class="fa fa-trash" style="font-size: 20px; color: red;"></i></a></li>

            </ul>
        </div>
    </div>

{% endfor %}

{% endblock %}
```

Figura 46. Bucle de elemento en HTML con jinja

Cada elemento de la cuadrícula de la página ficheros se genera en el bloque anterior de código. Este recoge un array de elementos al renderizar el componente y construye un bloque por cada elemento del array. Podemos ver el for que se usa para iterar en el listado de elementos:

```
{% for file in files %}
```

Figura 47. Atributo para declarar bucles en jinja

Por último, se muestra un ejemplo de cómo se renderiza esta vista con el array de ficheros obtenido de la base de datos:

```
@app.route('/files', methods=['GET', 'POST'])
@check_cookie
def files(session):
    if request.method == 'POST':
        pass
    else:
        files = File.query.filter_by(user=session.id).all()
        return render_template('files.html', files=files, session=session)
```

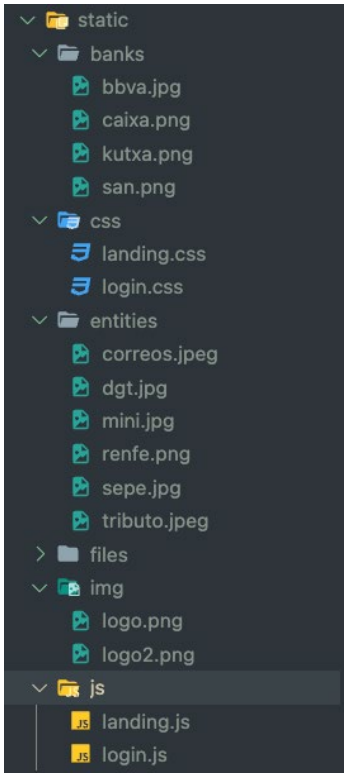
Figura 48. Ruta que renderiza una lista de objetos

Primero se extrae de la base de datos todos los ficheros del usuario actual y se le pasa tanto la sesión con el listado de los ficheros a la función “render_template”.

Todas las vistas de la aplicación funcionan con un sistema similar, modificando las variables de la plantilla base.

7.8.2 Estáticos

Los ficheros estáticos, como su nombre indica, no son modificado en tiempo de ejecución. Estos engloban los ficheros javascript, CSS e imágenes.



Imágenes de los bancos.

Hojas de estilo de la interfaz.

Imágenes de las entidades.

Imágenes de perfil de usuario.

Código javascript.

Figura 49. Jerarquía de los elementos estáticos de la aplicación.

8 Casos de uso

En este capítulo se describen los casos de uso final de la aplicación, es decir, la solución que ofrece a los usuarios y entidades que la utilicen.

Al suplir ciertas necesidades en función del tipo de cuenta con la que se inicie sesión, las distintas operativas se separan en:

- **Login con autenticación basada en web3.0, común a ambos usuarios.**
- **Casos de uso de un usuario particular.**
- **Casos de uso de una entidad.**

8.1 Login con autenticación basada en web3.0

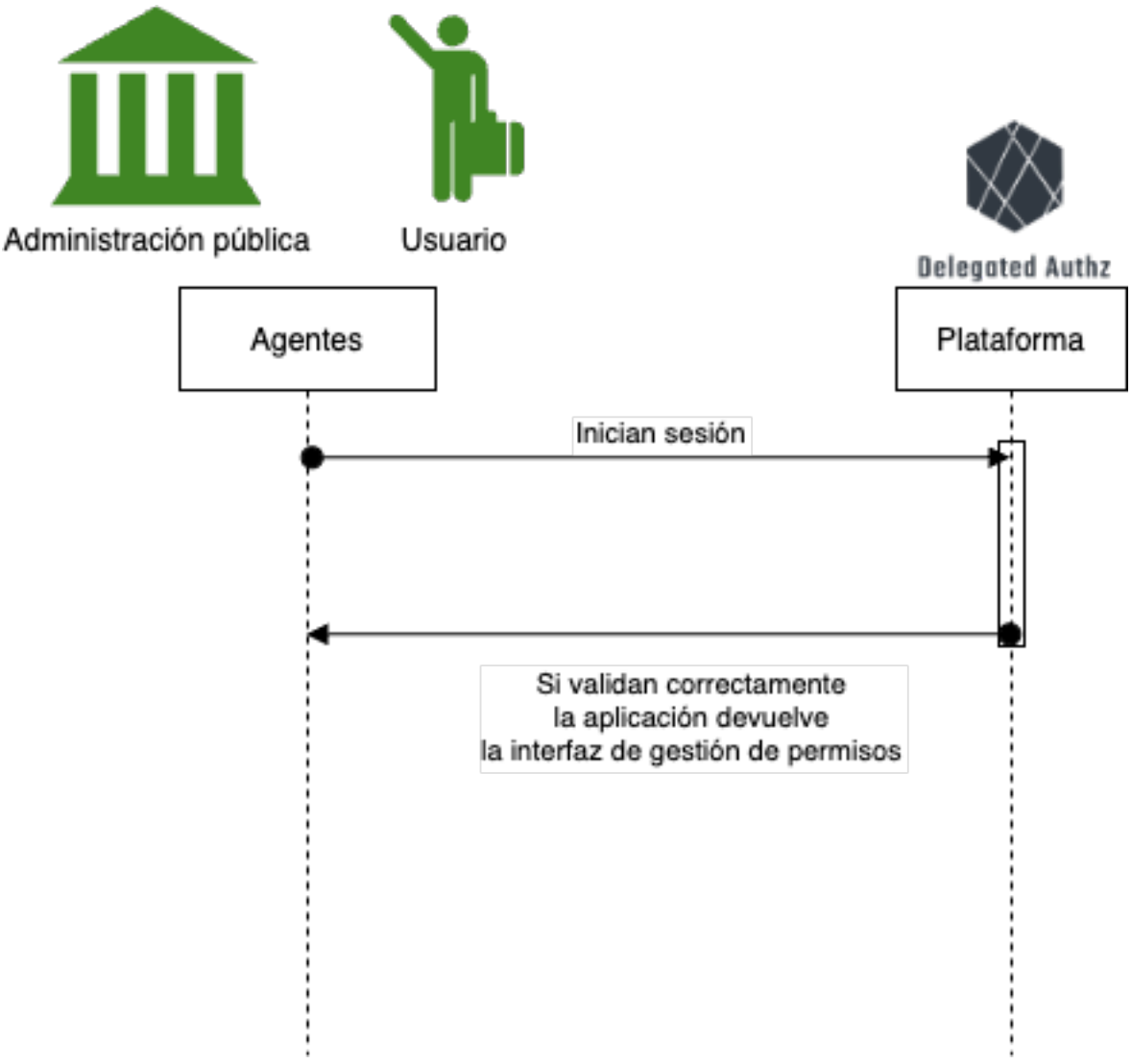


Figura 50. Diagrama de casos de uso común a todos los usuarios.

El sistema de login es común a cualquier tipo de usuario. Este utiliza la extensión metamask para albergar las carteras de Ethereum de los usuarios y las protege con una capa de autenticación extra.

A su vez, metamask valida contra la red de Ethereum las claves privadas de los usuarios para ver que las cuentas estén dadas de alta en la blockchain.

Esto permite que el sistema de autenticación, credenciales y datos personales, estén en propiedad del usuario o, al menos, en servicios de su confianza.

La aplicación ofrece una vista de inicio de sesión simple en la que el usuario no tiene que introducir ninguna credencial, al pulsar el botón de login, el navegador intentará buscar un servicio que soporte el método de autenticación web3.0.

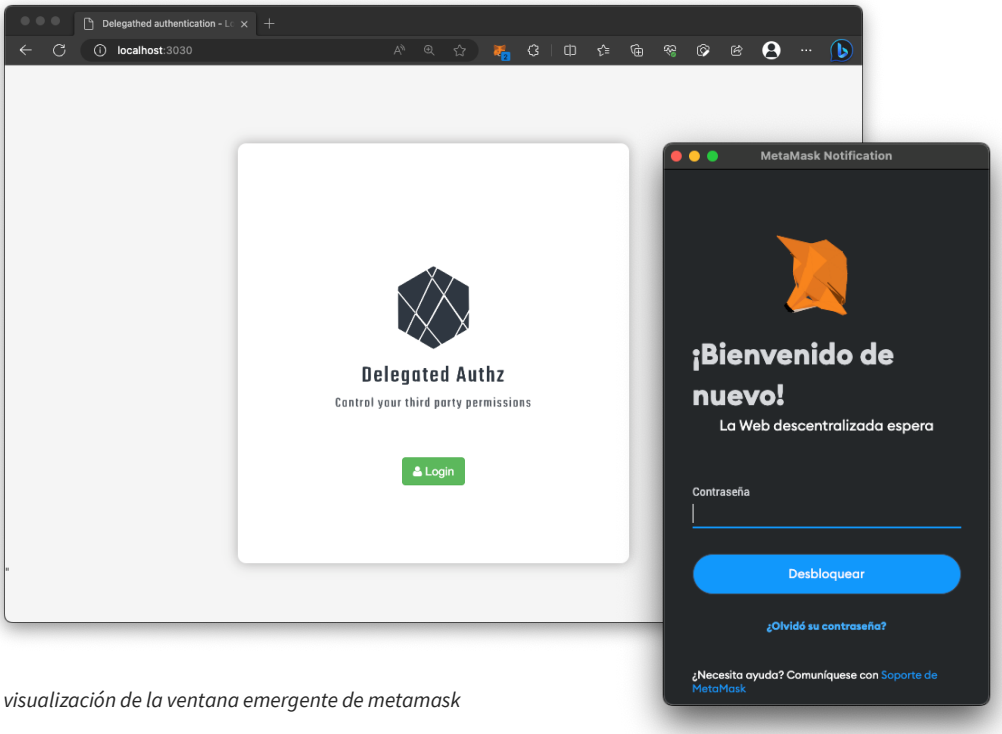


Figura 51. Ejemplo de la visualización de la ventana emergente de metamask

En este caso, la extensión de metamask detecta intento de inicio de sesión y despliega una ventana flotante permitiéndonos iniciar sesión en nuestra cartera de Ethereum.

Posteriormente, esta extensión valida la información y, si es correcta, permite al usuario entrar a la aplicación.

La aplicación soporta dos tipos de usuario:

- Usuario particular: Una persona o usuario individual que quiere gestionar los permisos a entidades terceras.
- Entidad: representa a una organización administrativa, en el contexto de este proyecto, a los diferentes administraciones o empresas públicas del país, agencia tributaria, SEPE, Renfe… etc.

Tanto un usuario como una entidad accederían a la aplicación de la misma manera y, en función del tipo de usuario, la aplicación permitiría unas opciones y otras.

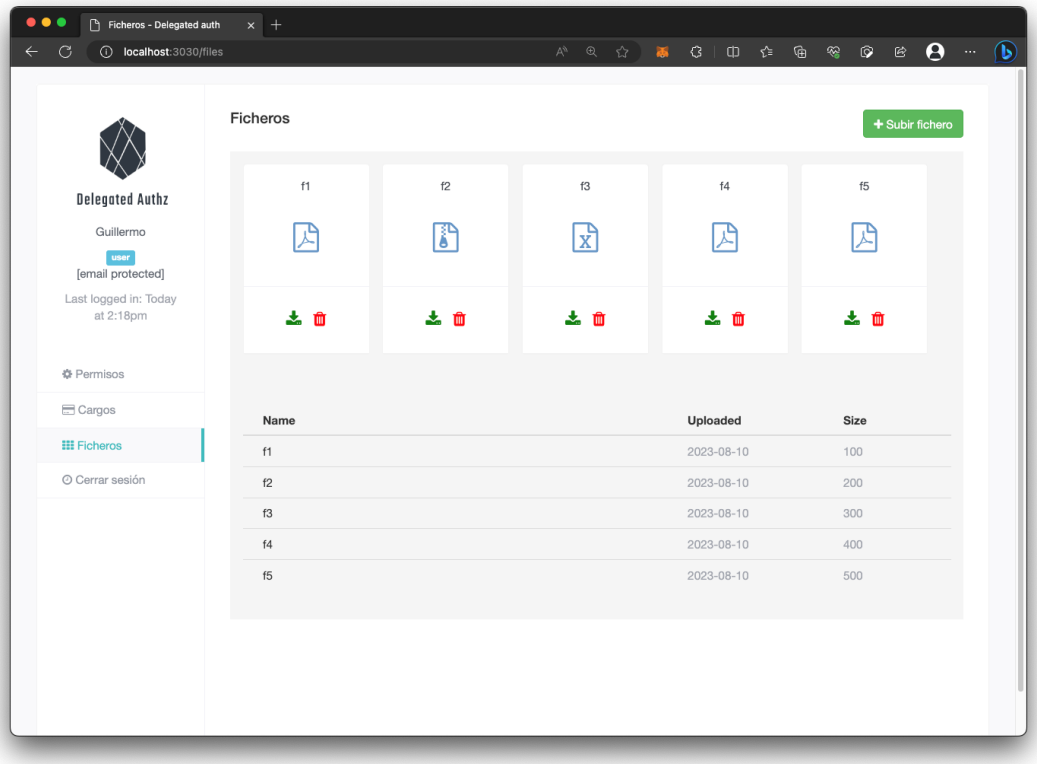


Figura 52. Vista inicial de la aplicación

Esta sería la primera ventana que un usuario, de cualquier tipo, vería al entrar a la aplicación.

8.1.1 Cierre de sesión

Un usuario puede cerrar, por seguridad, su sesión manualmente (aunque también se puede gestionar los tiempos de sesión desde metamask) pulsando sobre el botón “Cerrar sesión” en el menú lateral izquierdo.

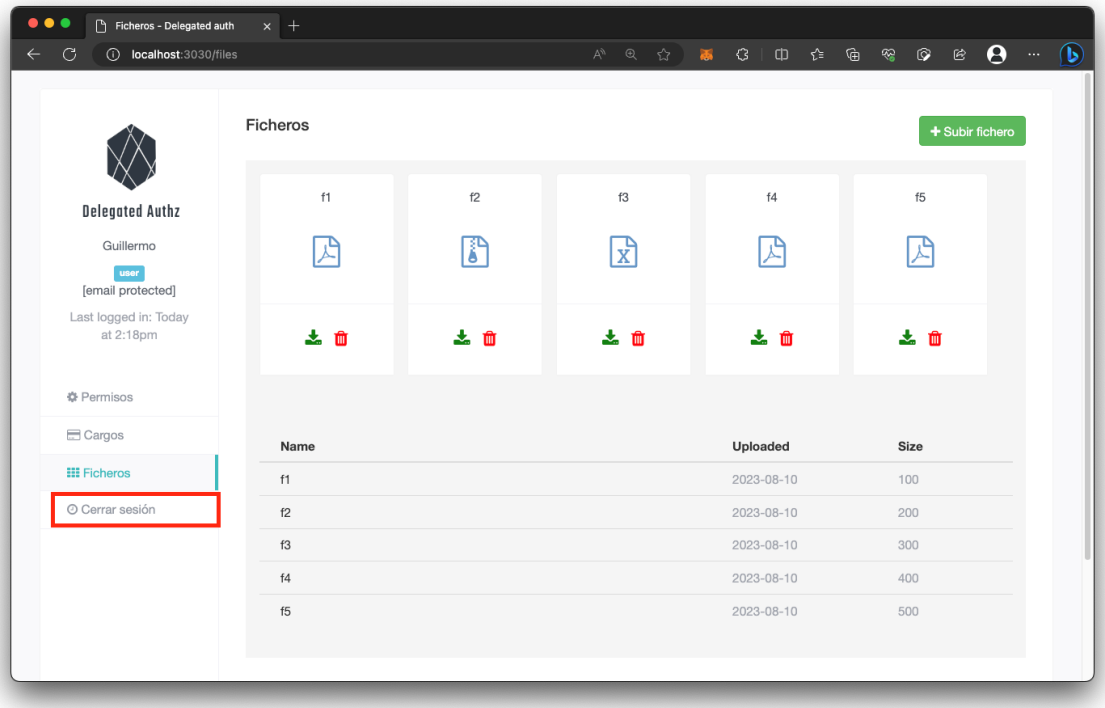


Figura 53. Cierre de sesión en la aplicación.

Esto nos devolvería automáticamente a la página de inicio de sesión y no nos permitiría volver a entrar hasta que volviéramos a autorizarnos con metamask.

8.2 Introducción a los casos de uso específicos

Los siguientes casos presentados en los puntos 8.3 y 8.3 son específicos de cada tipo de agentes.

Los agentes que utilizan la aplicación se dividen en usuarios y entidades. Los usuarios son los interesados finales de la solución, los que necesitan el control y la gestión de sus delegaciones de cobros mientras que, las entidades, consumirían la aplicación para enviar cobros (impuestos, multas, servicios) a los usuarios.

Esta relación entra las acciones de los usuarios y las entidades se detalla en el siguiente diagrama:

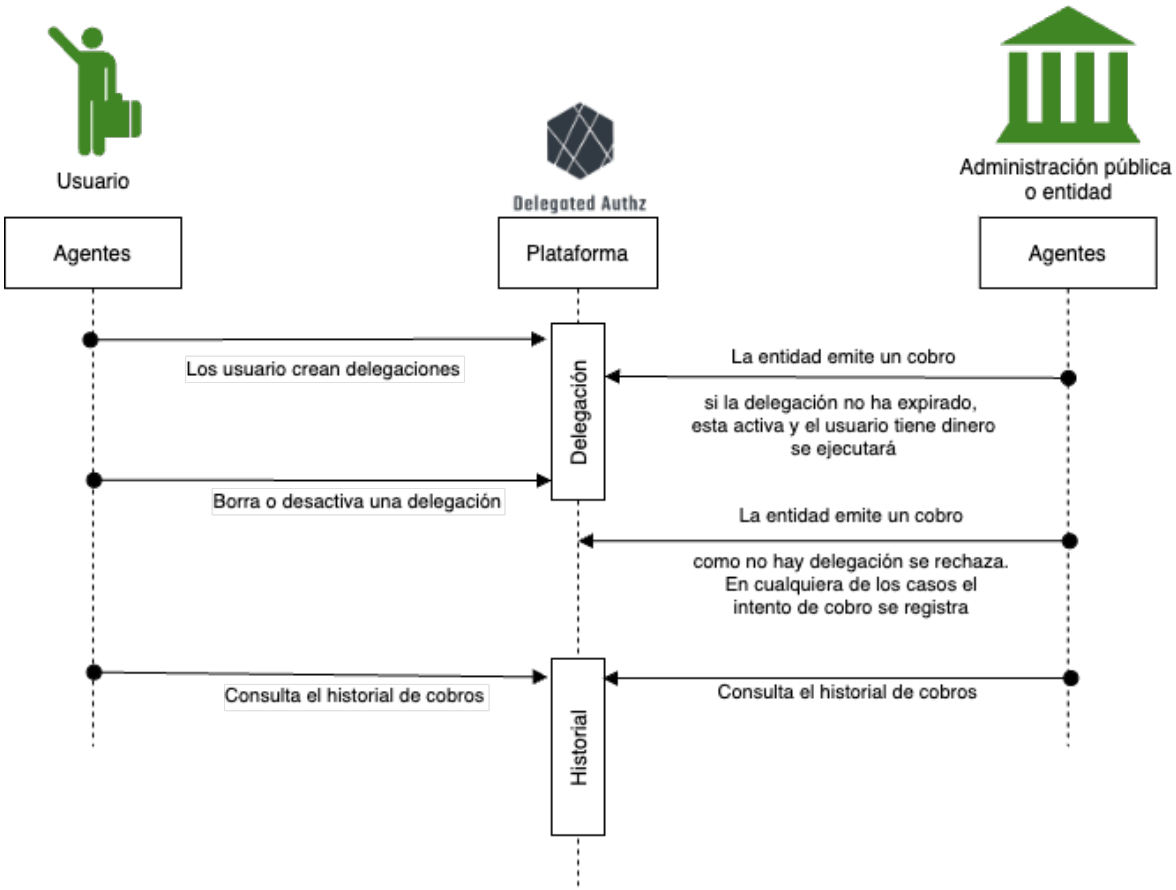


Figura 54. Resumen casos de uso específicos

En la figura anterior, se pueden observar los principales casos de uso de cada agente y su intersección en cuento a la delegación (la responsable de que un cobro se haga efectivo o no) y el historial de cada uno de ellos.

8.3 Casos de uso de un usuario particular

A continuación, se detallan las diferentes funcionalidades ofrecidas a un usuario particular. Principalmente se agrupan en la:

- **Gestión de autorizaciones.**
- **Gestión de entidades de cobro.**
- **Consulta de cargos recibidos.**

8.3.1 Gestión de autorizaciones

Las autorizaciones permiten crear un permiso, el cual tiene una fecha de expiración y se puede activar o desactivar, que nos permitirá decirle a una entidad si el cobro de un cargo se acepta o no se acepta.

Se puede acceder a la vista de autorizaciones con el botón “Permisos” en la barra lateral izquierda:

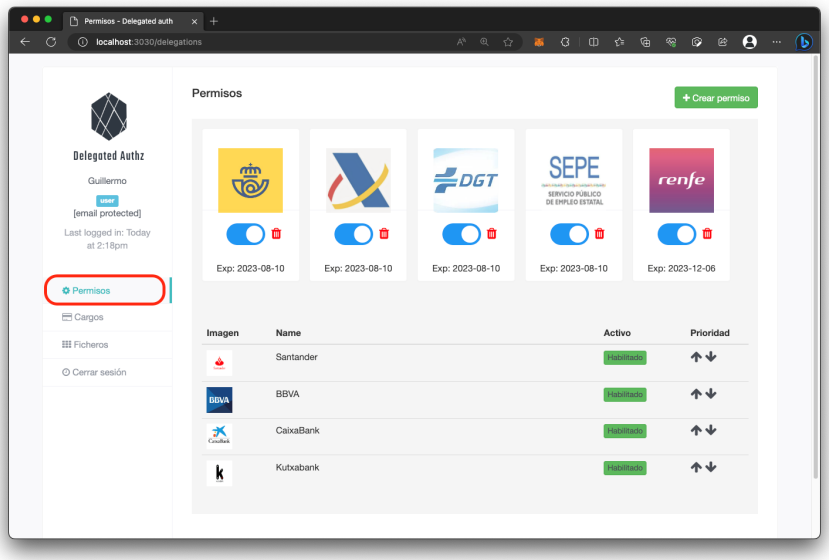


Figura 55. Acceso a la vista de permisos o delegaciones

La vista de autorizaciones o permisos se divide en, el bloque de autorizaciones en sí mismas (bloque 1) y las entidades de cobro (bloque 2), en el cual, se representan los diferentes bancos que tengamos configurados.

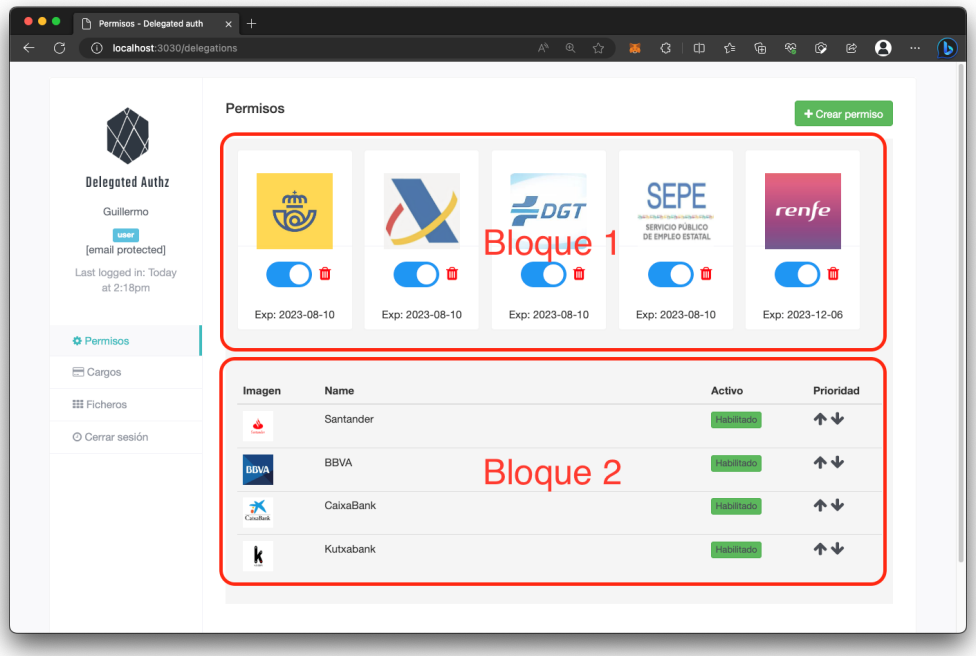


Figura 56. Detalle de la vista de permisos

8.3.1.1 Creación

Podemos crear nuevas autorizaciones pulsando en el botón “Crear permiso” en la sección de “Permisos”:

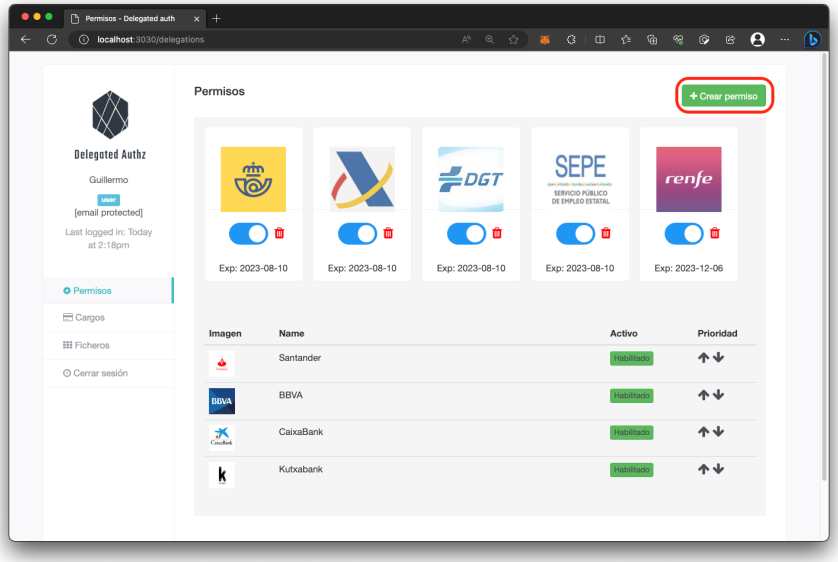


Figura 57. Botón de creación de permisos

Este nos cargará un formulario, el cual, nos permitirá elegir a la entidad a la que queremos darle el permiso para automatizar los cobros y la fecha de expiración del permiso. Ambos campos son obligatorios.

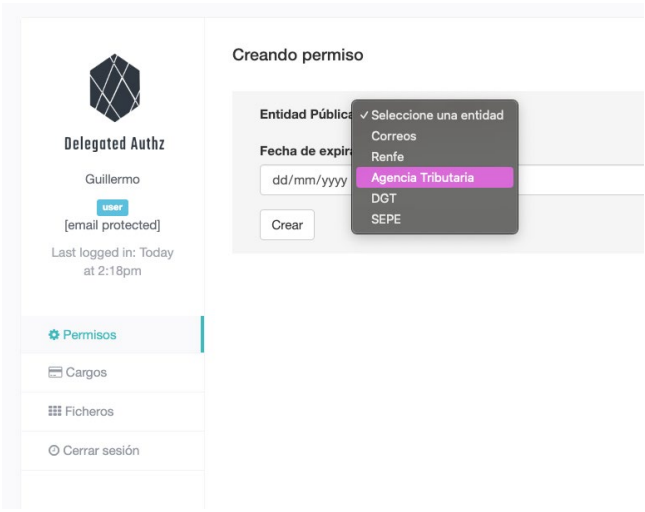


Figura 58. Formulario de creación de permisos

Cuando creamos un nuevo permiso o delegación, nos saldrá listado junto al resto en la vista general de permisos.

8.3.1.2 Borrado

Se puede eliminar cualquiera de los permisos otorgados utilizando el icono de borrado individual.

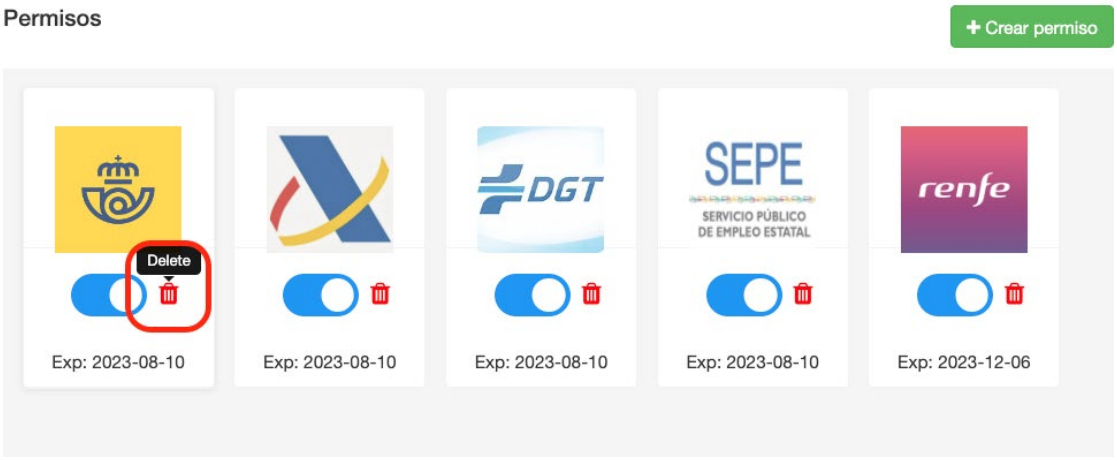


Figura 59. Borrado de delegaciones

La aplicación eliminaría el permiso automáticamente y recargaría el listado de delegaciones de forma transparente.

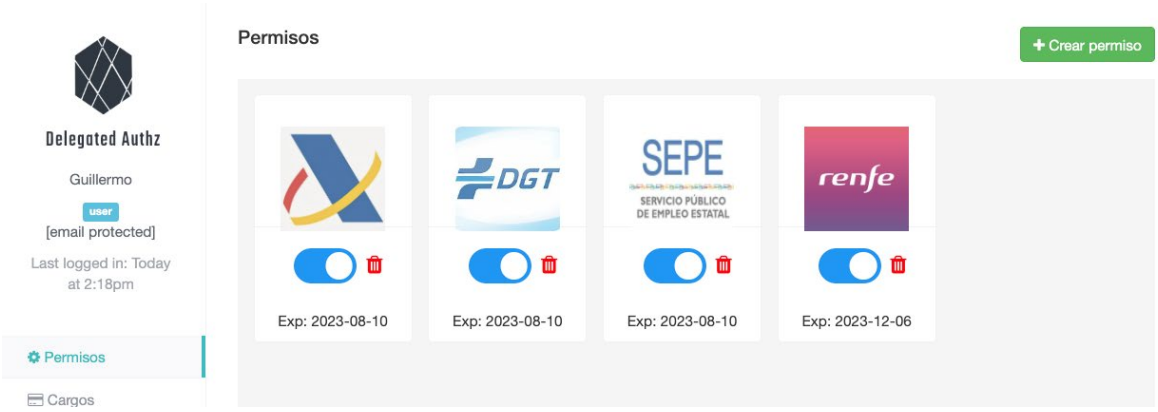


Figura 60. Comprobación del borrado de una delegación

8.3.1.3 Fecha de expiración

Como se ha descrito antes, cada delegación tiene una fecha de expiración asociada, la cual no es modificable ni por el usuario ni la entidad. Podríamos crear varias delegaciones o permisos y se respetaría siempre la de mayor expiración, permitiendo guardar un historial de delegaciones, aunque estas ya habrían expirado.

Esta fecha se representa individualmente en la vista de “Permisos” de la aplicación:

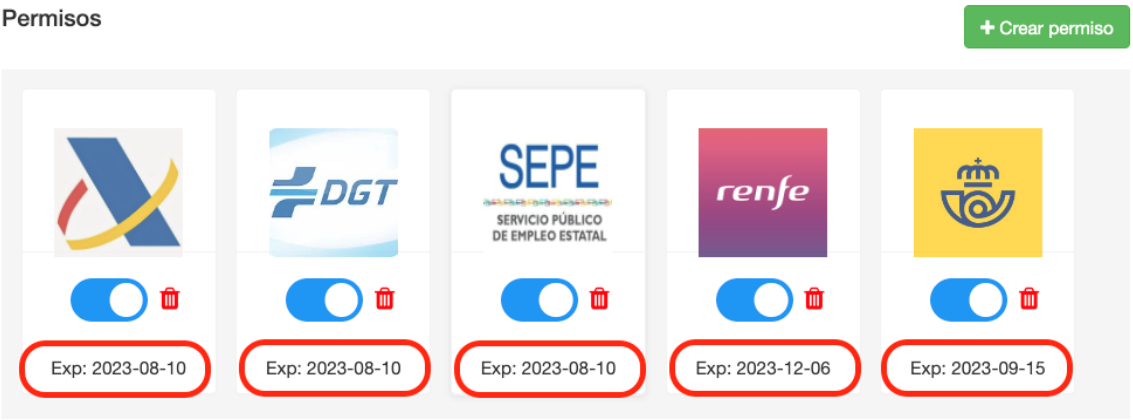


Figura 61. Vista de las fechas de expiración

8.3.1.4 Habilitar y deshabilitar

La aplicación permite activar y desactivar individualmente cualquiera de las delegaciones. Esto, de forma complementaria, nos permite crear franjas temporales en las que aceptar, o no hacerlo, cobros de cualquier de las administraciones sin tener que revocar el permiso.

Se puede utilizar esta funcionalidad con cada uno de los botones de tipo candado que se lista junto a cada permiso.

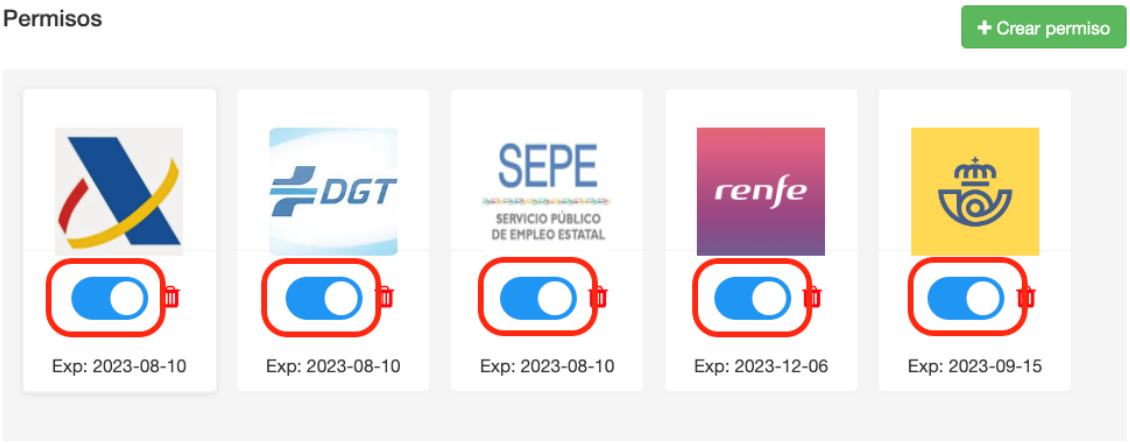


Figura 62. Botones para activar o desactivar delegaciones

Si desactivamos una delegación, se mostrará una animación y, sin recargar la página, la aplicación guardará el estado de esta delegación impidiendo cualquier cobro en tiempo real. Además, se mostraría en otro tono de color más neutro reflejando, en un vistazo, que permisos están activados y cuáles no.

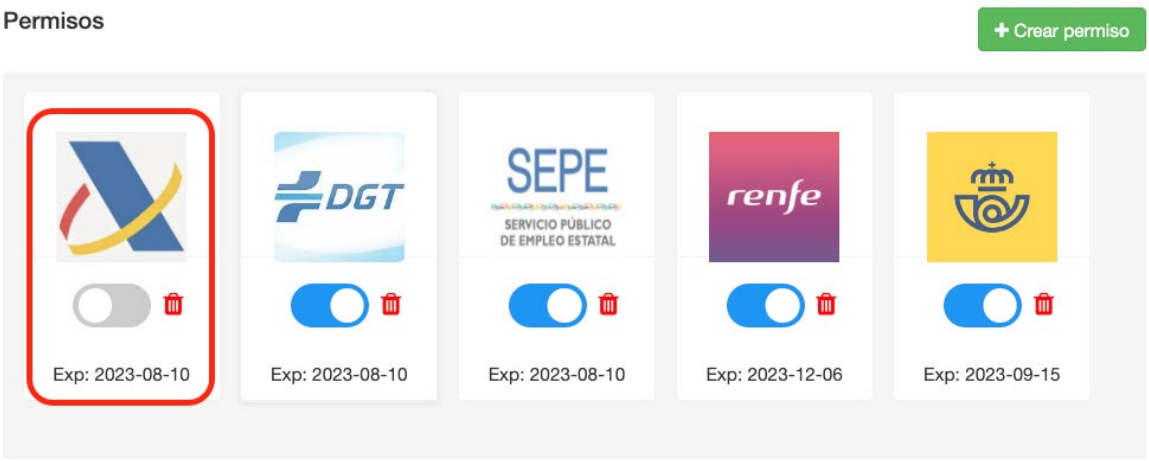


Figura 63. Desactivación de una delegación

8.3.2 Gestión de entidades de cobro

Las entidades de cobro hacen referencia a cualquier entidad que permita recibir los cargos que se efectúan al usuario. Lo normal es que sean bancos, aunque se deja la puerta abierta a cualquier entidad financiera que permitieran realizar la operación, por ejemplo, servicios de banca digital o incluso pagos en criptodivisas.

En el alcance actual del proyecto, estas entidades están preconfiguradas y se permiten realizar las distintas acciones:

8.3.2.1 Habilitar y deshabilitar

Se permite activar y desactivar cualquier de las entidades de cobro de forma individual. Esto permite al usuario evitar cobros en cuentas no deseadas, evitando descubiertos u otro tipo de imprevistos.

Para habilitar o deshabilitar cualquier entidad de cobro solo tenemos que pulsar sobre el estado, el cual hace también de botón, y automáticamente y sin recargar la web alternara el estado actual a su opuesto.

Figura 64. Habilitar una entidad bancaria

Por ejemplo, desactivando las entidades “Santander” y “CaixaBank” la aplicación se vería de la siguiente manera:

Figura 65. Vista de las entidades bancarias con ejemplos desactivados

Cuando se intenta hacer un cobro sobre las entidades de un usuario, se buscan primero las que estén activas y, en caso de haber varias, se respeta la prioridad seleccionada por el usuario. Si no existiera ninguna entidad activa todos los cobros se rechazarían automáticamente.
El funcionamiento de las prioridades se describe en el siguiente punto.

8.3.2.2 Prioridades

Aunque en un futuro se podría valorar distribuir un cobro entre varias entidades, el proyecto asocia un cargo o cobro a una entidad individual.

Como hemos visto en el punto anterior, estas se pueden activar o desactivar, y al recibir un cobro solo se tendrán en cuenta las entidades habilitadas.

Dentro de las entidades habilitadas, el usuario marca un orden de prioridades descrito en el orden en el que se listan en la vista de “Permisos”.

Un cobro se intentará ejecutar en la primera entidad habilitada de la lista, en el siguiente ejemplo se realizaría sobre el banco Santander:

Figura 66. Vista de las entidades bancarias

En la última columna de la tabla, tenemos los botones de priorizades, los cuales, nos permiten modificar las prioridades de cada entidad.

Cada flecha es un botón individual, los cuales, nos permitirían modificar el orden de la lista alternando las prioridades con el elemento directamente encima o debajo de la entidad que queremos modificar.

Por ejemplo, si quisiéramos modificar la prioridad de la entidad “Santander”, la cual está en primera posición, solo podríamos utilizar la flecha orientada hacia abajo para alterna la posición con el siguiente elemento de la lista.

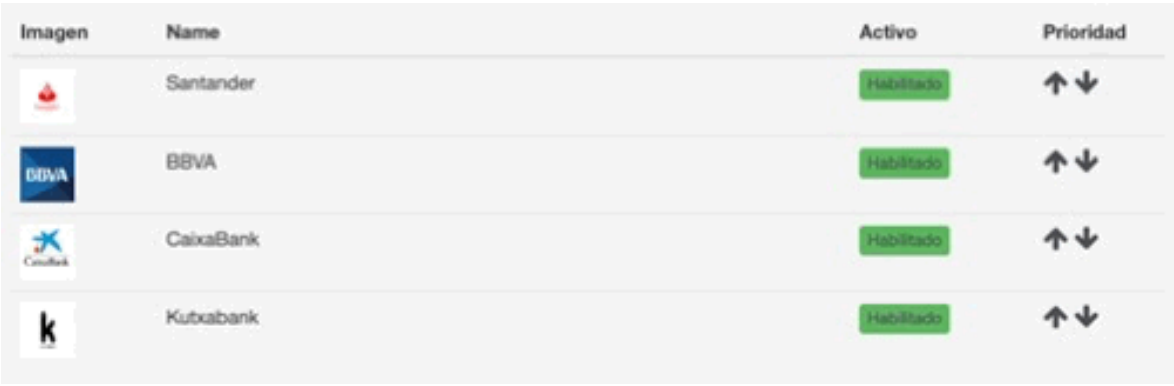


Figura 67. Gif del cambio de prioridades

8.3.3 Consulta de cargos recibidos

En la sección “Cargos”, un usuario puede consultar todos los cargos que ha recibido tanto si se han aceptado o rechazado. Por cada cargo, se lista la entidad que lo ha emitido, la entidad o banco donde se ha ejecutado el cargo, fecha, cantidad y, por último, descripción.

Esta sería la vista de un usuario con varios cargos:

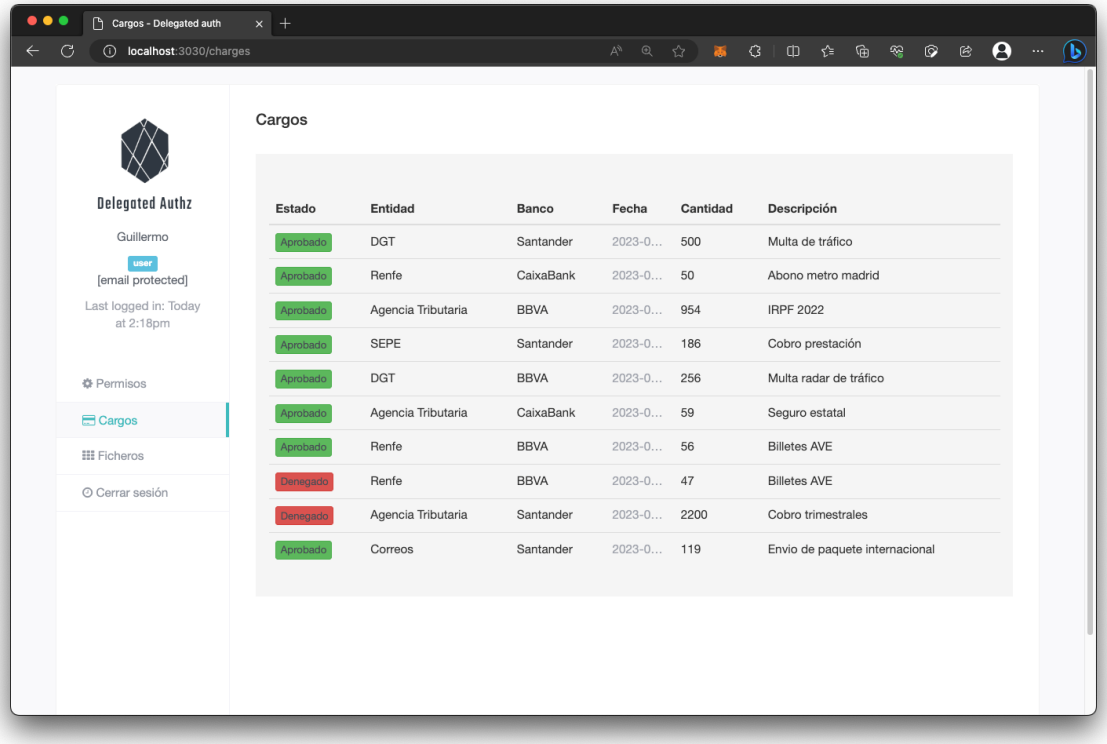


Figura 68. Vista de cargos recibidos

8.4 Casos de uso de una entidad

En el siguiente bloque se describen los casos de uso de una entidad. Aunque originalmente el alcance de la aplicación se centraba en la vista de un usuario particular, se comprendió indispensable ciertas vistas y funcionalidades desde la perspectiva de la entidad que emite un cobro.

Si iniciamos sesión con una entidad, podemos ver ligeras variaciones en la interfaz. En este caso, la “Agencia tributaria” no tiene ficheros; se muestra una etiqueta que señala el tipo de usuario de la aplicación, en este caso “entidad”; y no se muestra la opción de delegaciones.

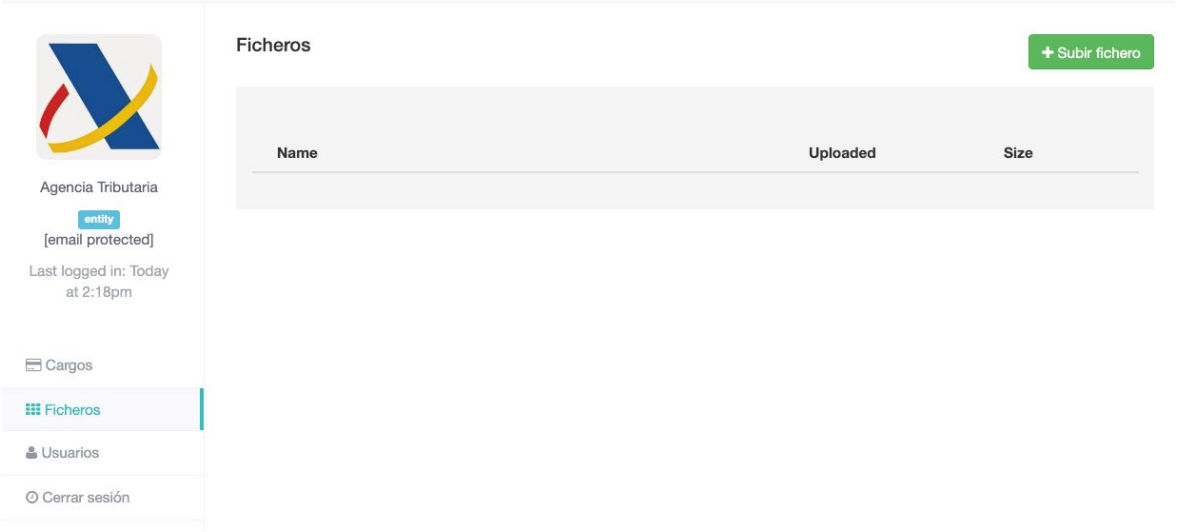


Figura 69. Vista inicial de una entidad

A cambio, la entidad tiene una nueva vista de usuarios, que muestra todos los usuarios que le han concedido permisos para ejercer cobros y, además, también tiene la capacidad de crearlos.

8.4.1 Consulta de cargos emitidos

En la vista de cargos, la entidad puede consultar todas las órdenes de cobro que ha emitido, así como los bancos que se han hecho responsables del importe y, por último, cantidades y detalles del cobro.

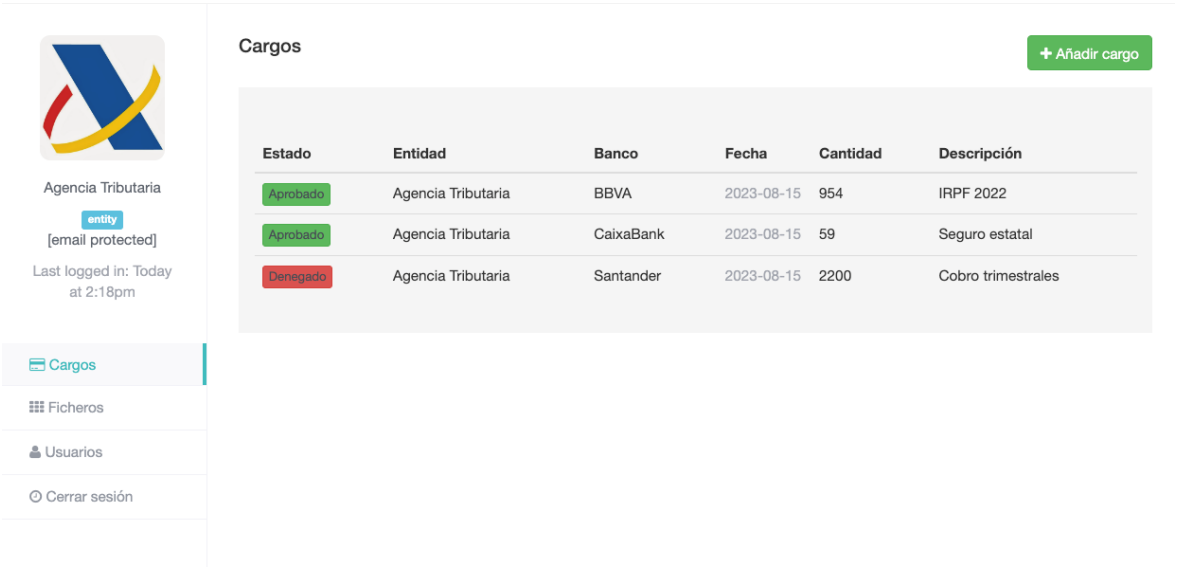


Figura 70. Consulta de los cargos emitidos por una entidad

Al ser una entidad, esta vista (similar a la del usuario particular) también cuenta con el botón “Añadir cargo”, funcionalidad que se describe más adelante en esta sección.

8.4.2 Consulta de usuarios con cobro delegado

Como entidad, es interesante conocer todos los usuarios que tienen activa una delegación de permisos. Esto permite tener un registro histórico de los cobros realizados, además de saber cuándo se retira esta delegación.

En la sección llamada “Usuarios”, tendremos esta vista listando todos loso usuarios con delegación activa para la entidad en cuestión. Se puede ver la dirección de la cartera de Ethereum junto con un botón de acción para realizar un cargo al usuario en cuestión.

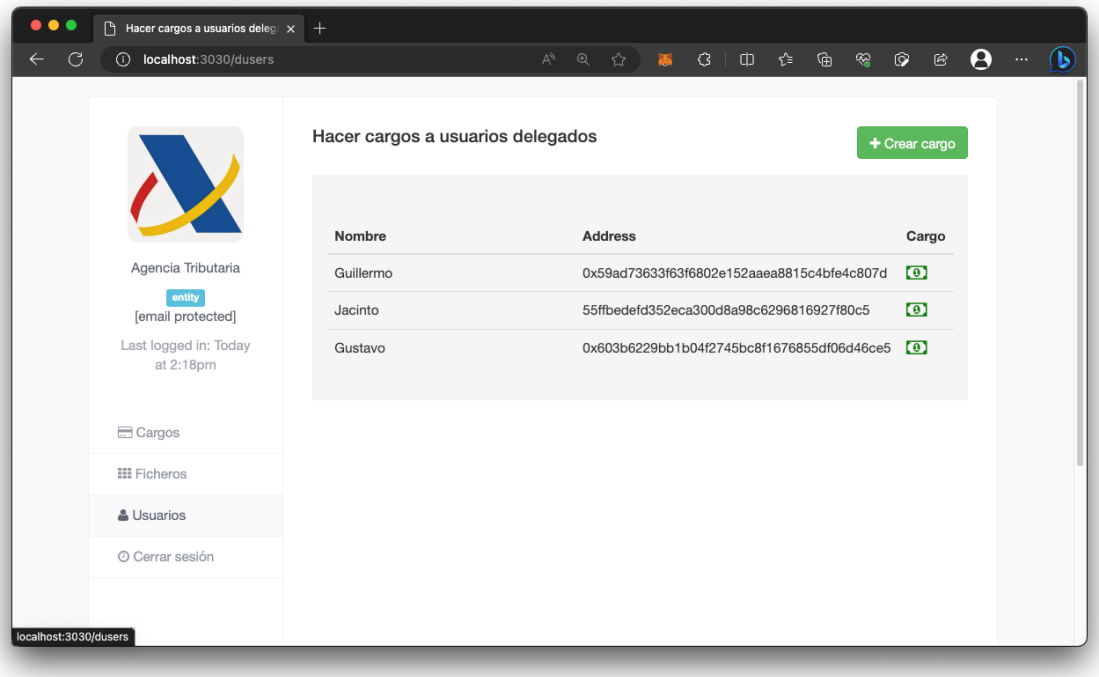


Figura 71. Vista de los usuarios que han delegado permisos a la entidad que usa la aplicación

8.4.3 Creación de cargos

Finalmente, y como máximo propósito de una entidad, la creación de cargos genera una orden del cobro de un importe a un usuario específico.

Para acceder a la vista de creación de cargos, se puede acceder a través del botón “Crear cargo” en las vistas de “Cargos” y “Usuarios” o en los botones individuales del listado de usuarios.

Esta es la vista de creación de cargos, cuenta con un formulario que nos permite seleccionar el usuario, la cantidad y una descripción.

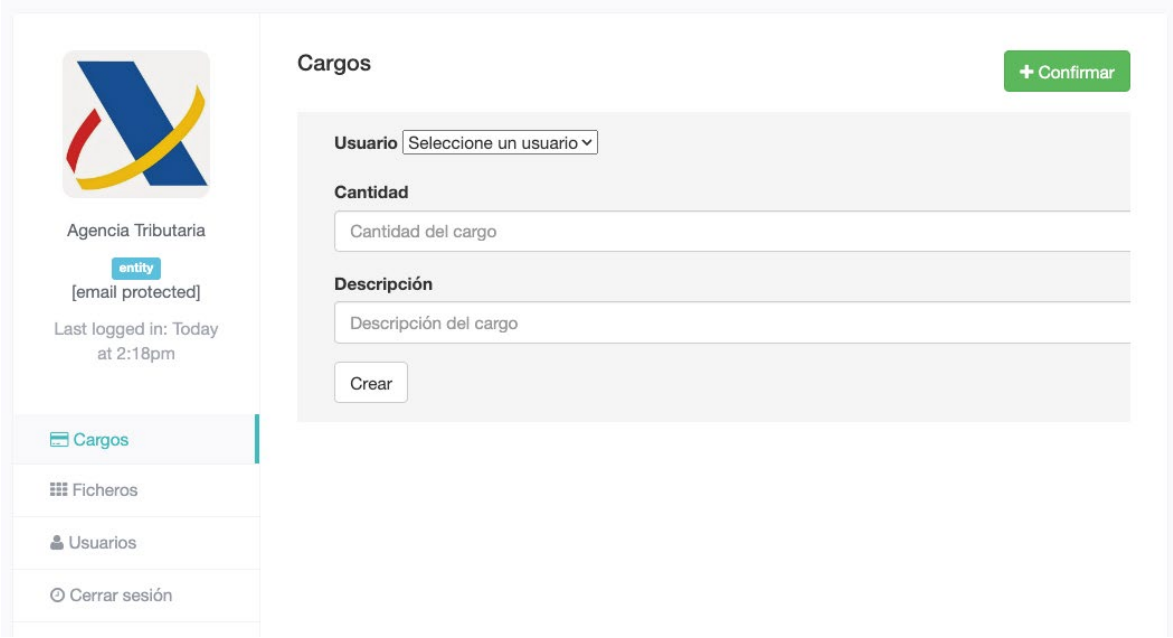


Figura 72. Formulario de creación de cobros

Vamos a simular la creación de un cargo. Se selecciona un usuario del listado desplegable, se rellena la cantidad y la descripción. Finalmente, se pulsa el botón confirmar para ejecutar la orden.

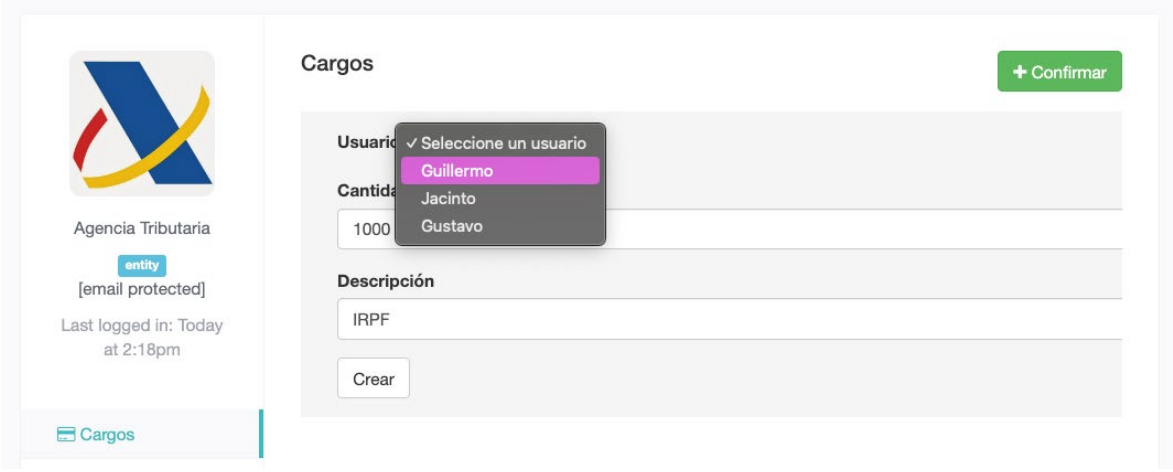


Figura 73. Desplegable de selección de usuarios con permiso de cobro activo

Refrescando otros puntos, y complementando el actual, una orden de cargo puede aceptarse o rechazarse.

El cargo se aceptaría si se cumplen todos los siguientes requisitos:

- El usuario tiene un permiso de delegación activo.

- El permiso de delegación no ha expirado.
- El usuario tiene al menos un banco configurado y habilitado.

Si alguno de los anteriores puntos no se cumple, se rechazaría el cargo, aunque, en cualquier caso, aparecería listado en el registro de cargos para que ambas partes estuvieran informadas de la imposibilidad de haberlo efectuado.

9 Análisis de Resultados

El análisis de resultados se ha agrupado en base a los objetivos planteados en el capítulo 3.2.1, los objetivos específicos 3.2.2 y los casos de uso más importantes (detallados en el capítulo 8).

Este análisis consiste en una enumeración de los diferentes casos de uso y objetivos en los que individualmente, se realiza una reflexión de como se ha completado y abordado la tarea intentando hacer también hincapié en sus debilidades o mejoras a futuro (todos ellas detalladas en el capítulo 10.2).

9.1 Objetivos Generales

1. **Crear un sistema automático, transparente y fiable**:
   - **Resultado**: La implementación del blockchain ha proporcionado una transparencia y fiabilidad inherentes en las transacciones. La automatización se ha logrado mediante la centralización de procesos de pago, pero aún hay margen de mejora, como lo demuestra el trabajo futuro relacionado con la migración total a Ethereum (capítulo 10.2.1).

2. **Respeto a la privacidad y propiedad de la información del usuario**:
   - **Resultado**: El uso del blockchain junto con la autenticación web3.0 ha establecido un sistema robusto que respeta la privacidad del usuario. Sin embargo, la autenticidad y propiedad de la información es un trabajo en progreso, con miras a una integración total con Ethereum. Además, actualmente se tiene que apoyar sí o sí en servicios de terceros como metamask, en un futuro se podría trabajar en ofrecer más opciones o incluso proporcionar al usuario la capacidad de implementar su propio sistema (o gestionarse el mismo la clave privada de su cartera) para respetar al máximo la privacidad del mismo.

9.2 Objetivos Específicos

1. **Automatización de la gestión de cobros de las entidades públicas**:
   - **Resultado**: Se ha establecido un panel y conjunto de programas que facilitan la automatización de los procesos. La centralización del proceso de pago ha demostrado ser efectiva, aunque se busca una mayor eficiencia a través de futuras integraciones.

2. **Implementación de un sistema de autorizaciones para cargos**:
   - **Resultado**: Se ha implementado un sistema preliminar de autorizaciones que permite a los usuarios definir la duración de los permisos. Aunque es efectivo, se busca mejorar su flexibilidad y robustez en futuras iteraciones.

3. **Adaptabilidad y Escalabilidad**:
   - **Resultado**: Si bien el sistema actual funciona para las necesidades inmediatas, los trabajos futuros, como la migración a Ethereum y la integración con bancos de terceros, indican un impulso hacia una mayor escalabilidad.

9.3 Casos de Uso

1. **Pago de impuestos y multas**:
   - **Resultado**: Los usuarios ahora pueden gestionar y recibir sus impuestos y multas desde un solo lugar, reduciendo la necesidad de interacciones manuales y múltiples transacciones bancarias con diversas entidades. Ejemplo común, el pago de un impuesto a través de la agencia tributaria o el pago de una multa de tráfico se realizan en diferentes sitios (en muchas circunstancias ni siquiera son automáticas). El mayor reto para que este punto funcionara en un entorno real, sería la integración con las diversas administraciones públicas, por lo que, lo ideal es que este proyecto naciera o fuera construido por la misma administración como complemento y evolución de sus sistemas tradicionales.

2. **Gestión de permisos y autorizaciones**:
   - **Resultado**: El sistema permite a los usuarios gestionar permisos para cargos de diferentes administraciones públicas. Si bien esto ha mejorado la experiencia del usuario, el trabajo futuro indica que hay áreas de mejora en términos de funcionalidad y facilidad de uso, incluyendo aplicaciones móviles (más utilizadas que las aplicaciones web dado el extendido uso del smartphone) o permitiendo participar a más agentes en la plataforma. Ideas detalladas en los capítulos 10.2.2 y 10.2.4.

10 Conclusiones y Trabajos Futuros

10.1 Conclusiones

El proyecto que se ha llevado a cabo ha sido un extenso recorrido por la integración de herramientas innovadoras con sistemas más convencionales. Una de las decisiones clave fue el uso de una red de blockchain local, que no solo proporcionó un ambiente seguro para experimentar y afinar detalles, sino que también sentó una base firme para la futura escalabilidad del proyecto (capítulo 10.2, trabajos futuros).

Un aspecto destacado fue la introducción de un nuevo método de inicio de sesión (capítulo 5, sistema desarrollado), que no solo incrementa los niveles de seguridad, sino que también muestra una nueva manera de interactuar en el ámbito digital. A través de este proceso, se evidencia el potencial de nuevas formas de verificación que van más allá de las contraseñas tradicionales.

El equilibrio entre una experiencia de usuario atractiva y un sistema interno robusto es esencial. Por un lado, se garantiza que los usuarios tengan una interacción fluida y sencilla con la plataforma. Por otro lado, se asegura que, detrás de escena, cada acción y petición estuviera bien fundamentada y protegida. Esta dualidad resalta la importancia de combinar facilidad de uso con seguridad (capítulo 7, implementación).

El uso de herramientas modernas para gestionar la información muestra cuán esencial es tener un sistema que pueda adaptarse y crecer según las necesidades del proyecto. Se logra un diseño claro y eficiente de la información, lo que permite una implementación ágil y, al mismo tiempo, sólida.

Este proyecto tiene una fuerte investigación en varias áreas: tecnología, analizando en profundidad las diversas herramientas y estándares; sociedad, al tener en cuenta las preocupaciones actuales de las personas en materia administrativa, **destacando la transparencia, el permiso explícito y la gestión soberana de la información**; y, por último, los problemas actuales de la administración pública a nivel de agilidad y automatización.

El proyecto cumple todos los objetivos propuestos desde su inicio, crear un sistema de autorización de cobros delegada, utilizando estándares basados en web3 y Ethereum. No lejos de estar terminado, al menos una versión comercializable, ya representa una idea sólida y funcional para dar un enfoque a la gestión de los datos y permisos en internet, devolviendo la propiedad de estos al usuario.

Se espera que este proyecto pueda tener continuidad, planteando las siguientes mejoras a futuro y consolidando la idea inicial del proyecto.

10.2 Trabajos futuros

Aunque en el estado actual, la aplicación, podría operar en un entorno real con alguna limitación, no deja de ser una prueba de concepto que necesita muchas mejorar.

Estas mejoras se detallan a continuación:

10.2.1 Migración a un funcionamiento total con Ethereum:

A medida que la tecnología blockchain avanza y se establece en diversos ámbitos, se prevé que Ethereum, una de las plataformas líderes en este campo, jugará un papel esencial en muchos sistemas. El proyecto actualmente puede estar aprovechando parcialmente las capacidades de Ethereum, pero una migración completa permitirá:

- **Optimización de Costos**: Al aprovechar las características nativas de Ethereum, el proyecto puede reducir costos asociados con soluciones intermedias o compatibilidades. Esto permitiría prescindir de las bases de datos relacionales.

- **Mayor Seguridad**: Ethereum, al ser una red descentralizada, ofrece un nivel de seguridad robusto contra fraudes y ataques. Además de que todos los agentes verificar la información (de forma anónima) guardada en la red.

- **Integración con Contratos Inteligentes**: Permitirá automatizar procesos, garantizando transparencia y autonomía en las transacciones.

En este supuesto, **el módulo de autenticación seguiría funcionando de la misma manera** ya que actualmente las cuentas las validad sobre la red de Ethereum, el resto de la solución habría que reestructurarla significativamente a nivel de lógica del dato para poder prescindir de la base de datos tradicional.

A continuación, se detalla los pasos que habría que llevar a cabo para completar esta migración:

10.2.1.1 Pasos de la migración

1. **Análisis de la Infraestructura Actual**: Revisar el sistema existente para identificar los componentes que se pueden trasladar directamente a Ethereum y aquellos que necesitan ser reescritos o adaptados. Concretamente, se prescindiría de las bases de datos relacionales (MySQL y SQLite) para la implementación de la solución.

2. **Evaluación de Costos**: Ethereum requiere gas (una forma de pago) para realizar transacciones y ejecutar contratos inteligentes. Es fundamental tener una idea clara de los costos involucrados para no incurrir en gastos inesperados. Habría que analizar que agente asumiría el coste para garantizar la viabilidad de la solución, inicialmente, tendría que ser el usuario, enfocando el producto como una solución más segura a los métodos actuales. A medio o largo plazo, lo ideal sería que la propia administración pública apoyara soluciones de este tipo como una ventaja a la ciudadanía que integrara en sus procesos o soluciones actuales.

3. **Desarrollo de Contratos Inteligentes**: Con Solidity, el lenguaje de programación principal de Ethereum, se diseñarán y desarrollarán contratos inteligentes para llevar a cabo las funcionalidades específicas del proyecto en la cadena de bloques. Como la escritura y consulta de cobros, las delegaciones (y expiración de las mismas) y los agentes que operan sobre la aplicación.

4. **Pruebas en Redes de Prueba**: Antes de desplegar en la red principal, es esencial probar en redes como Ganache para garantizar que todo funcione como se espera y no incurrir en el coste (gas) que tendría cada operación. Actualmente, se ha utilizado para la parte de autenticación y se podría volver a utilizar para simular la integración total.

5. **Despliegue y Monitorización**: Una vez que todo esté listo y probado, se puede desplegar en la red principal de Ethereum. Es vital monitorizar el rendimiento, los costos y cualquier posible error o vulnerabilidades que pueda surgir en la plataforma. Teniendo en cuenta que se van a manejar datos financieros de los usuarios, habría que tener especial enfoque en la seguridad, manteniendo el sistema siempre lo más actualizado posible.

10.2.1.2 Desafíos de Ethereum

El principal reto de esta migración recae en su lenguaje de programación, solidity. Es un lenguaje relativamente nuevo, poco conocido y que puede suponer los principales problemas:

Madurez Relativa: Aunque Solidity es el lenguaje principal para desarrollar en Ethereum, todavía se considera relativamente nuevo y está en constante evolución, lo que puede presentar desafíos en términos de compatibilidad y estabilidad.

Errores Costosos: Un error en un contrato inteligente puede resultar en pérdidas significativas de dinero o en vulnerabilidades de seguridad. No hay opción de "deshacer" en la cadena de bloques.

Optimización del Gas: Escribir código eficiente en Solidity es esencial para minimizar los costos de gas. Esto a veces puede ir en contra de las prácticas de codificación tradicionales.

Entorno Limitado: Los contratos inteligentes en Ethereum tienen limitaciones en términos de capacidad de procesamiento y almacenamiento.

Interacción con Otros Sistemas: Integrar contratos inteligentes con sistemas y plataformas externas puede ser complicado y requiere soluciones como oráculos para llevar información fuera de la cadena al contrato.

Aprendizaje y Recursos: Aunque la comunidad Ethereum es amplia y activa, todavía hay una curva de aprendizaje asociada con Solidity y la programación de contratos inteligentes. Es esencial estar al día con las mejores prácticas y cambios en el lenguaje y la plataforma.

10.2.2 Creación de un sistema de integraciones de entidades y bancos de terceros

Uno de los mayores desafíos en cualquier proyecto financiero es la integración con diferentes entidades y bancos. Establecer contacto y acuerdos individualmente con entidades financieras puede ser una tarea titánica e inalcanzable para un proyecto de esta envergadura.

Establecer un sistema robusto y flexible para estas integraciones permitirá que cualquier banco o sistema financiero puede integrarse a través del API de la aplicación y, en un inicio, hasta despertar interés, trabajar con pasarelas de pago de terceros.

Aunque las comisiones iniciales pudieran ser un bloqueante para muchos usuarios, a día de hoy, existen muchas empresas con Redsys, Paypal, Stride… etc., que por una mínima comisión podrían operar contra cualquier entidad bancaria o tarjeta bancaria y solucionar el problema inicial de las integraciones.

10.2.3 Pago con criptodivisas

El mundo está avanzando rápidamente hacia la aceptación de criptodivisas no solo como una inversión, sino también como un medio de pago válido. Actualmente la solución solo permite configurar bancos tradicionales como agentes (asociados a usuarios) que reciben los cobros de la administración pública. Integrar esta opción en el proyecto ofrecerá:

- **Diversidad de Opciones de Pago**: Atraerá a un segmento de la población que posee criptomonedas y busca formas de gastarlas.

- **Reducción de Barreras Transaccionales**: Las criptodivisas eliminan la necesidad de intermediarios bancarios y ofrecen transacciones más rápidas y, a menudo, más baratas.

- **Innovación y Competitividad**: Situará al proyecto a la vanguardia de las tendencias tecnológicas y financieras, destacándolo como una solución moderna y actualizada.

10.2.4 Desarrollo de Aplicaciones Móviles

A medida que la movilidad se ha convertido en una parte esencial de nuestro día a día, tener una presencia en dispositivos móviles es vital para cualquier plataforma en línea. La adaptación de la solución a plataformas móviles traerá consigo diversas ventajas y desafíos:

- **Accesibilidad y Conveniencia**: Una aplicación móvil permitirá a los usuarios acceder a los servicios del proyecto en cualquier momento y lugar, facilitando la gestión de sus impuestos y multas en movimiento.

- **Notificaciones en Tiempo Real**: Las aplicaciones móviles pueden enviar notificaciones push a los usuarios, informándoles sobre eventos importantes en los cobros, fechas de expiración próxima para las delegaciones de permisos… etc.

- **Funcionalidades Nativas**: Las aplicaciones móviles pueden aprovechar las características y funcionalidades nativas de los dispositivos, como escáneres biométricos, cámaras y sistemas de geolocalización, para mejorar la experiencia del usuario y aumentar la eficiencia.

- **Interfaz de Usuario Adaptada**: Una aplicación móvil requerirá un diseño y una interfaz de usuario adaptados específicamente para pantallas pequeñas, garantizando que los usuarios tengan una experiencia fluida y agradable que a día de hoy la página web no ofrece en dispositivos con pantallas pequeñas.

- **Integración con Wallets Móviles**: Con el crecimiento de las criptomonedas, muchas personas usan wallets móviles para gestionar sus activos digitales. Integrar la plataforma con estas wallets facilitaría los pagos y transacciones, por ejemplo, servicios como Apple Pay, Google Pay o Samsung Pay para facilitar al usuario la configuración de entidades a las que asociar los cobros automáticos de la aplicación.

Teniendo en cuenta que la solución se ha diseñado de forma modular, las aplicaciones móviles (o cualquier nuevo tipo de consumir la aplicación) solo tendría que hacer llamadas al módulo de autenticación y el backend para funcionar de la misma manera que la aplicación web.

Estas propuestas para trabajos futuros, si se implementan, no solo mejorarán la funcionalidad del sistema, sino que también lo posicionarán como una solución competitiva en el mercado actual, dada la baja competencia y el auge de este sector.

11 Bibliografía

Blockchain y Criptomonedas:

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
- Tapscott, D., & Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin.
- Zohar, A. (2015). *Bitcoin: under the hood*. Communications of the ACM, 58(9), 104–113.

Gestión soberana:

- Allen, C. (2016). *The Path to Self-Sovereign Identity*. Coindesk.
- Tobin, A., & Reed, D. (2016). The Inversion of the Self-Sovereign Internet and the Rise of the Sovrin Network. Sovrin Foundation White Paper.
- Baars, D. (2016). Towards self-sovereign identity using blockchain technology. University of Twente.
- Sabah, A. (2017). *Decentralized blockchain-based electronic marketplaces*. Communications of the ACM, 61(1), 78-84.
- Dunphy, P., & Petitcolas, F. A. (2018). *A first look at identity management schemes on the blockchain*. IEEE Security & Privacy, 16(4), 20-29.
- Brodersen, C., Pöhls, H. C., & Strufe, T. (2017). *Definition and Implementation of a SAML-X. 509 Usage Profile for the Datastore-trusted Data Exchange Protocol*. Proceedings of the 32nd International Conference on ICT Systems Security and Privacy Protection.
- Hardman, D. (2018). *Sovrin's Token: A Stake in the Ground*. Sovrin Foundation White Paper.
- Preukschat, A., & Reed, D. (Eds.). (2020). Self-Sovereign Identity: Decentralized digital identity and verifiable credentials. Manning Publications Co.
- Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, S., Gonthier, G., Kobeissi, N., ... & Zaliva, V. (2016). *Formal verification of smart contracts: Short paper*. Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security.

Sistemas Distribuidos:

- Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall.
- Cachin, C., & Vukolić, M. (2017). *Blockchain consensus protocols in the wild*. arXiv preprint arXiv:1707.01873.

Autenticación Web 3.0:

- Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann.
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The Semantic Web*. Scientific American, 284(5), 34-43.

Gestión de Permisos y Cuentas Bancarias:

- Cameron, K., & Jones, M. B. (2016). *Design Rationale behind the Identity Metasystem Architecture*. Microsoft Corporation.
- Frazier, J. (2019). *The Future of Banking: The Role of Information Technology*. Information Systems Education Journal, 17(3), 56.

ODS y Desarrollo Sostenible:

- United Nations. (2015). *Transforming our world: The 2030 agenda for sustainable development*. United Nations.
- Sachs, J. D. (2015). *The age of sustainable development*. Columbia University Press.

Automatización de Pagos y Finanzas:

- Mulligan, C., & Olsson, U. (2013). *Architecting a global real-time payments network*. Computer Networks, 57(17), 3386-3400.
- Casey, M. J., & Wong, P. (2018). *The truth machine: The blockchain and the future of everything*. St. Martin's Press.

Administración Pública y Tecnologías Emergentes:

- Mergel, I. (2016). *Public administration in the age of big data*. In *Big Data in Public Affairs* (pp. 1-18). Routledge.
- Dunleavy, P., & Hood, C. (2017). *From old public administration to new public management*. Public Money & Management, 37(6), 365-372.

12 ANEXO I: Alineación de proyecto con los ODS

Los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas establecen una agenda global para abordar los desafíos sociales, económicos y ambientales que enfrenta nuestro mundo. Estos objetivos nos brindan una hoja de ruta para lograr un desarrollo sostenible y crear un futuro mejor.

En este contexto, el proyecto de implementación de un Sistema Distribuido para automatizar pagos de impuestos y multas, y mejorar la gestión de permisos y cuentas bancarias, se alinea de manera destacada con el ODS número 16: "Paz, Justicia e Instituciones Sólidas". Además, también presenta conexiones significativas con otros objetivos de desarrollo sostenible.

Objetivo Principal: Objetivo de Desarrollo Sostenible 16 - Paz, Justicia e Instituciones Sólidas
El ODS 16 busca promover sociedades pacíficas, inclusivas y justas, y garantizar el acceso a la justicia para todos. El proyecto de implementación del Sistema Distribuido contribuye a este objetivo de varias maneras:

- **Acceso a la justicia**: Al automatizar el pago de impuestos y multas, el sistema garantiza que los ciudadanos tengan un acceso más fácil y eficiente a los servicios gubernamentales. Esto reduce las barreras burocráticas y facilita la participación ciudadana en el cumplimiento de sus obligaciones legales.

- **Transparencia y rendición de cuentas**: La integración con una red blockchain proporciona una capa adicional de transparencia en el proceso de pago. Todas las transacciones se registran de forma inmutable y se vuelven transparentes, lo que promueve la rendición de cuentas de las entidades públicas y fortalece la confianza de los ciudadanos en el sistema.

- **Reducción de la corrupción**: Al automatizar los pagos y eliminar intermediarios innecesarios, se reducen las oportunidades de corrupción. La descentralización y la gestión soberana basada en blockchain ayudan a garantizar la integridad de las transacciones y a minimizar el riesgo de malversación de fondos.

- **Participación ciudadana**: La plataforma, al ser transparente y accesible, puede fomentar una mayor participación ciudadana en la toma de decisiones y en la supervisión de la administración de impuestos y multas.

- **Protección de Datos Sensibles**: El uso de tecnologías como blockchain garantiza la privacidad y seguridad de los datos de los usuarios, fortaleciendo así la seguridad y el respeto a los derechos individuales.

12.1 Conexiones con otros Objetivos de Desarrollo Sostenible

Además de su alineación con el ODS 16, el proyecto también presenta conexiones significativas con otros objetivos de desarrollo sostenible:

- **Objetivo 9 - Industria, Innovación e Infraestructura**: La implementación de un sistema distribuido implica la integración de tecnologías innovadoras como blockchain y autenticación Web 3.0. Estas tecnologías avanzadas fomentan la creación de infraestructuras resilientes, promueven la innovación y fortalecen las capacidades tecnológicas del país.

- **Objetivo 11 - Ciudades y Comunidades Sostenibles**: Al automatizar los pagos de impuestos y multas, se mejora la eficiencia y la calidad de los servicios públicos para los ciudadanos, contribuyendo a crear ciudades y comunidades más sostenibles.

- **Objetivo 17 - Alianzas para lograr los objetivos**: El proyecto involucra la colaboración entre entidades públicas, instituciones financieras y proveedores de servicios tecnológicos, lo que es fundamental para su éxito.

- **Objetivo 8 - Trabajo decente y crecimiento económico**: Al garantizar una gestión fiscal eficiente, se crea un entorno propicio para la inversión y el crecimiento económico. La implementación de tecnologías avanzadas también puede generar empleos en sectores tecnológicos y de innovación.

- **Objetivo 12 - Producción y consumo responsables**: El proyecto promueve un consumo y producción más responsables al garantizar que las empresas cumplan con sus obligaciones fiscales de manera justa y transparente.

El proyecto de implementación de un Sistema Distribuido para automatizar pagos de impuestos y multas, y mejorar la gestión de permisos y cuentas bancarias, se alinea de manera sólida con el Objetivo de Desarrollo Sostenible 16: Paz, Justicia e Instituciones Sólidas. Al proporcionar un acceso más fácil a la justicia, fomentar la transparencia y la rendición de cuentas, y reducir la corrupción, el proyecto contribuye a fortalecer las instituciones y promover sociedades más justas y pacíficas.

Además, el proyecto presenta conexiones relevantes con otros objetivos de desarrollo sostenible, como la promoción de la innovación y la infraestructura, la construcción de ciudades y comunidades sostenibles, y la promoción de alianzas para lograr los objetivos. Estas conexiones resaltan la importancia y el potencial del proyecto para generar un impacto positivo en múltiples dimensiones del desarrollo sostenible.
A través de la implementación de este sistema distribuido, se abre la puerta hacia una administración fiscal más eficiente, transparente y justa, sentando las bases para un futuro sostenible y equitativo.

13 ANEXO II: Glosario

ORM (Object-Relational Mapping): es una técnica de programación que permite mapear objetos de una aplicación a las tablas de una base de datos relacional. Proporciona una abstracción de la base de datos y permite interactuar con ella utilizando objetos y métodos en lugar de escribir consultas SQL directamente.

PoS (Proof-of-Stake): Prueba de Participación. Es un algoritmo de consenso utilizado en blockchain que permite a los validadores de la red crear nuevos bloques y confirmar transacciones en función de la cantidad de criptomoneda que poseen y están dispuestos a "apostar" como garantía. A diferencia de PoW, que se basa en la capacidad de cálculo, PoS se basa en la tenencia de activos y tiene como objetivo reducir el consumo de energía.

PoW (Proof-of-Work): Prueba de Trabajo. Es un algoritmo de consenso utilizado en blockchain en el que los participantes, llamados mineros, deben realizar cálculos complejos para resolver un problema criptográfico. El primer minero en resolver el problema agrega el siguiente bloque a la cadena y es recompensado con nuevas criptomonedas. PoW es conocido por su alta demanda de energía.

ICO (Initial Coin Offering): Oferta Inicial de Moneda. Es un método de recaudación de fondos utilizado en el espacio de las criptomonedas y blockchain. En una ICO, una empresa o proyecto emite nuevas criptomonedas o tokens y las vende a inversores a cambio de criptomonedas más establecidas, como Bitcoin o Ethereum. Los inversores obtienen estos tokens como una inversión en el proyecto futuro.

DeFi (Decentralized Finance): Finanzas Descentralizadas. Se refiere a un conjunto de aplicaciones financieras construidas en blockchain que buscan eliminar intermediarios tradicionales en los servicios financieros. Estas aplicaciones incluyen préstamos, intercambios, seguros y más, y operan en entornos descentralizados y transparentes.

IDE (Integrated Development Environment): Entorno de Desarrollo Integrado. Es un software que proporciona herramientas y características para facilitar el desarrollo de aplicaciones. Un IDE generalmente incluye un editor de código, depurador, compilador, gestión de proyectos y otras utilidades que ayudan a los desarrolladores a escribir, probar y depurar su código de manera más eficiente.

DBMS (Database Management System): Sistema de Gestión de Base de Datos. Es un software que permite a los usuarios almacenar, gestionar, acceder y manipular datos en una base de datos. Proporciona interfaces para crear, modificar y consultar bases de datos, así como garantizar la integridad y seguridad de los datos.

RDBMS (Relational Database Management System): Sistema de Gestión de Base de Datos Relacional. Es un tipo de DBMS que se basa en el modelo relacional, donde los datos se almacenan en tablas con filas y columnas. Los RDBMS utilizan el lenguaje SQL (Structured Query Language) para realizar operaciones como consultas, inserciones, actualizaciones y eliminaciones en los datos almacenados.

API (Application Programming Interface): Interfaz de Programación de Aplicaciones. Es un conjunto de reglas y protocolos que permite que diferentes aplicaciones o sistemas se comuniquen entre sí. Las API definen cómo las distintas funciones y datos pueden ser solicitados y utilizados por otras aplicaciones.

Token: En el contexto de blockchain y criptomonedas, un token es una unidad digital que representa un valor o activo en una red blockchain. Los tokens pueden representar monedas, activos digitales, derechos de acceso u otras formas de valor.

Smart Contract: Contrato Inteligente. Es un código de programación autoejecutable que se ejecuta automáticamente cuando se cumplen ciertas condiciones predefinidas. Los smart contracts se ejecutan en una blockchain y permiten acuerdos y transacciones confiables sin intermediarios.

DApp (Decentralized Application): Aplicación Descentralizada. Es una aplicación que funciona en una red descentralizada de blockchain en lugar de depender de servidores centralizados. Las DApps utilizan contratos inteligentes y protocolos de blockchain para operar de manera autónoma.

Nonce: Número Aleatorio Único. En blockchain, un nonce es un número que se utiliza en el proceso de minería de PoW para encontrar un hash válido que cumpla con ciertas condiciones. Los mineros prueban diferentes valores de nonce para encontrar el hash correcto y crear un nuevo bloque.

Fork: Horquilla. En blockchain, un fork se produce cuando una cadena de bloques se divide en dos cadenas separadas debido a cambios en el protocolo o desacuerdos en la comunidad. Los forks pueden ser hard forks (cambios drásticos en el protocolo) o soft forks (cambios compatibles con versiones anteriores).

Consensus Algorithm: Algoritmo de Consenso. Es el método utilizado en una red blockchain para llegar a un acuerdo sobre el estado de la cadena y validar transacciones. Algunos ejemplos son PoW (Prueba de Trabajo) y PoS (Prueba de Participación).

Decryption: Desencriptación. Es el proceso de convertir datos cifrados nuevamente en su forma original legible. Requiere el uso de una clave de descifrado para revertir el proceso de cifrado y obtener acceso a los datos originales.

Encryption: Encriptación. Es el proceso de convertir datos legibles en un formato cifrado utilizando algoritmos y claves de cifrado. La encriptación se utiliza para proteger la confidencialidad de los datos y evitar accesos no autorizados.

Node: Nodo. En el contexto de blockchain, un nodo es un dispositivo o computadora que participa en la red y mantiene una copia completa de la cadena de bloques. Los nodos verifican y validan las transacciones y mantienen la integridad de la red.

Immutable: Inmutable. Se refiere a la propiedad de que los datos almacenados en una blockchain no pueden ser modificados ni eliminados una vez que se han registrado en un bloque. Esto contribuye a la seguridad y confiabilidad de la información en la cadena de bloques.

Gas: En blockchain, el gas es una unidad de medida utilizada para cuantificar el costo de ejecutar transacciones y contratos inteligentes. Cada operación en la red requiere una cantidad específica de gas para su ejecución y los usuarios pagan en función de la cantidad de gas utilizado.

Immutable Ledger: Registro Inmutable. Es un registro o libro mayor que no puede ser alterado o modificado una vez que la información ha sido registrada en él. Las blockchains ofrecen registros inmutables debido a la naturaleza de la tecnología de cadena de bloques.

Decentralization: Descentralización. Es el principio de distribuir el control y la toma de decisiones en una red o sistema en lugar de tener una única entidad o autoridad central. Las blockchains buscan la descentralización para evitar puntos únicos de falla y aumentar la seguridad.

Hash Function: Función de Hash. Es un algoritmo matemático que toma una entrada (como un archivo o dato) y produce una cadena de caracteres de longitud fija que parece aleatoria. Las funciones de hash se utilizan en blockchain para crear identificadores únicos y para garantizar la integridad de los datos.

Public Key: Clave Pública. En criptografía de clave pública, una clave pública es un valor que se utiliza para cifrar los datos y verificar firmas digitales. Puede ser compartida públicamente y se utiliza para cifrar mensajes que solo el titular de la clave privada correspondiente puede descifrar.

Private Key: Clave Privada. En criptografía de clave pública, una clave privada es un valor secreto que se utiliza para descifrar mensajes cifrados con la clave pública correspondiente y para firmar digitalmente datos. La seguridad de la clave privada es esencial para la seguridad de las transacciones en blockchain.

Double Spending: Doble Gasto. Es un problema en sistemas digitales donde un activo digital se gasta más de una vez. Las blockchains resuelven el problema del doble gasto al garantizar que las transacciones sean únicas y verificables, evitando así la posibilidad de gastar la misma moneda dos veces.

Mining: Minería. Es el proceso de validar y registrar transacciones en una blockchain mediante el uso de poder de cómputo para resolver problemas matemáticos. En PoW, los mineros compiten para encontrar el hash correcto y agregar un nuevo bloque a la cadena.

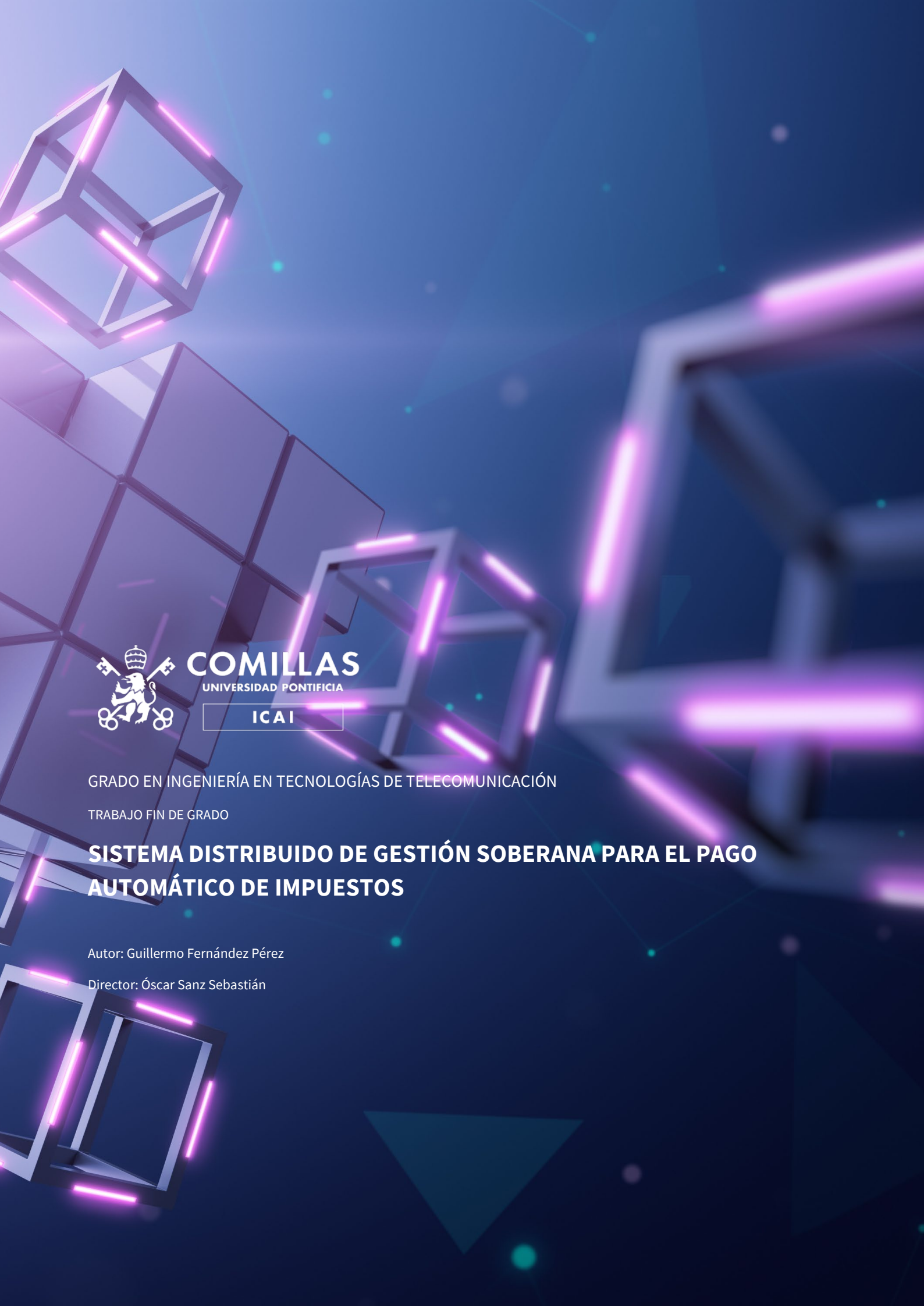
Wallet: Cartera. En el contexto de criptomonedas, una cartera es un software o dispositivo que permite a los usuarios almacenar, enviar y recibir activos digitales, así como gestionar sus claves privadas. Las carteras pueden ser en línea, móviles, de escritorio o hardware.

Fiat Currency: Moneda Fiat. Es una moneda emitida por un gobierno central y respaldada por su autoridad. Ejemplos de monedas fiat incluyen el dólar estadounidense, el euro y el yen japonés. Las criptomonedas a menudo se comparan con las monedas fiat en términos de valor y uso.

Immutable Contract: Contrato Inmutable. En el contexto de contratos inteligentes, un contrato inmutable es un contrato que no puede ser modificado después de su despliegue en la cadena de bloques. Los términos y condiciones del contrato están codificados en el contrato mismo y no pueden ser alterados.

Gas Limit: Límite de Gas. Es la cantidad máxima de gas que un usuario está dispuesto a gastar en una transacción o contrato inteligente en una blockchain. El límite de gas ayuda a controlar los costos y garantizar que una transacción no consuma más recursos de los necesarios.

CRUD: es un acrónimo que se refiere a las cuatro operaciones básicas que se pueden realizar sobre los datos persistentes en sistemas de bases de datos y otras aplicaciones de almacenamiento. Estas operaciones son crear (create), leer (read), actualizar (update) y eliminar (delete).



COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA DISTRIBUIDO DE GESTIÓN SOBERANA PARA EL PAGO AUTOMÁTICO DE IMPUESTOS

Autor: Guillermo Fernández Pérez

Director: Óscar Sanz Sebastián