Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Sistema de Verificación de Credenciales Profesionales Basado en Blockchain Privado

Autor
Pedro R. Cuevas Olarte

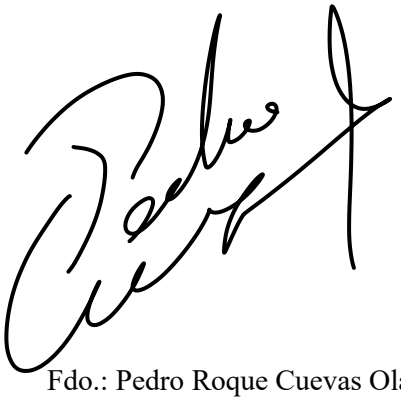
Director
Luis Francisco Sánchez Merchante

Madrid
Mayo 2023

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Sistema de Verificación de Credenciales Profesionales Basado en Blockchain Privado

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2022/23 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.: Pedro Roque Cuevas Olarte Fecha: 29/08/2023

Autoriza la entrega del proyecto

Luis Francisco Sánchez Merchante
2023.08.29 07:40:17 +02'00'

Fdo.: Luis Francisco Sánchez Merchante Fecha: 29/08/2023

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Sistema de Verificación de Credenciales Profesionales Basado en Blockchain Privado

Autor
Pedro R. Cuevas Olarte

Director
Luis Francisco Sánchez Merchante

Madrid
Mayo 2023

SISTEMA DE VERIFICACIÓN DE CREDENCIALES PROFESIONALES BASADO EN BLOCKCHAIN PRIVADO

Autor: Cuevas Olarte, Pedro Roque.

Director: Sánchez Merchante, Luis Francisco.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

La creciente digitalización de los registros académicos y la necesidad de garantizar su autenticidad ha llevado a la exploración de nuevas tecnologías que ofrezcan seguridad y eficiencia. En este contexto, el presente Trabajo de Fin de Grado aborda el desafío de desarrollar un entorno que facilite la acreditación de credenciales académicas utilizando una red blockchain pública permisionada, una tecnología emergente que ofrece inmutabilidad y transparencia en los registros.

El enfoque principal del proyecto fue la creación e implementación de un smart contract en la Red B de Alastria. Se escogió esta red debido a sus características únicas y su adaptabilidad para manejar datos sensibles como las credenciales académicas. El diseño del sistema también incluyó un frontend y un plugin para Moodle, permitiendo una interfaz amigable y familiar para los usuarios. Por último, se desarrolló un backend utilizando Django y SQLite para extender las funcionalidades del sistema y garantizar una interacción fluida.

A nivel técnico, el smart contract se diseñó y desplegó inicialmente en una red Hyperledger Besu local, antes de su implementación final en la Red B. Las pruebas en la API RESTful se llevaron a cabo utilizando Postman, mientras que el frontend se construyó con MDB y React. El plugin de Moodle, por su parte, se implementó en una instancia local de Moodle en XAMPP.

Los resultados obtenidos representan un avance significativo en el ámbito de las acreditaciones académicas digitales. Se logró construir un sistema robusto que permite a los usuarios registrar y validar información, facilitando así el proceso de acreditación y autenticación de documentos académicos en entornos digitales.

Palabras clave: blockchain, Moodle plugin, acreditaciones académicas, Hyperledger Besu, Alastria

Introducción

En el mundo globalizado actual, la transparencia y autenticidad de las credenciales educativas y profesionales son cruciales. La tecnología blockchain ofrece una respuesta a esta demanda. Proporcionando un registro inmutable y transparente [1], las blockchains públicas permisionadas en particular, presentan una solución prometedora para autenticar credenciales, fusionando seguridad con innovación tecnológica [2].

Por lo tanto, nuestro objetivo es generar un entorno completo que permita a usuarios e instituciones acreditar credenciales académicas en una red blockchain privada permisionada.

Metodología

Planteamiento del proyecto

En este trabajo se busca diseñar y crear un sistema que permita mejorar la accesibilidad, seguridad y transparencia para acreditar credenciales académicas.

Para ello se desarrollará un smart contract, que será desplegado en la Red B de Alastria. Este permitirá a los usuarios almacenar información suficiente como para acreditar la autenticidad de una acreditación académica y una serie de metadatos para mejorar la transparencia y confiabilidad del sistema.

Para el desarrollo se seguirá un proceso incremental, desplegando y testeando cada elemento de la arquitectura de manera independiente hasta que el sistema funcione de manera cohesionada. Se comenzará desarrollando el smart contract. Cuando este funcione en una red blockchain local, se pasará al desarrollo del resto de elementos en paralelo al despliegue definitivo del contrato en la Red B de Alastria.

Diseño de arquitectura

El sistema desarrollado cuenta con una arquitectura compuesta por componentes bien separados entre sí:

- Smart contract desplegado en red B de Alastria, encargado de registrar y permitir consultas de las acreditaciones académicas
- Plugin de Moodle en un despliegue local de Moodle, que permite al usuario interactuar con el smart contract
- Frontend: frontend desarrollado en React, que permite al usuario interactuar con el smart contract y con el backend
- Backend: backend Django, que añade funcionalidades a la web para mejorar la experiencia de usuario

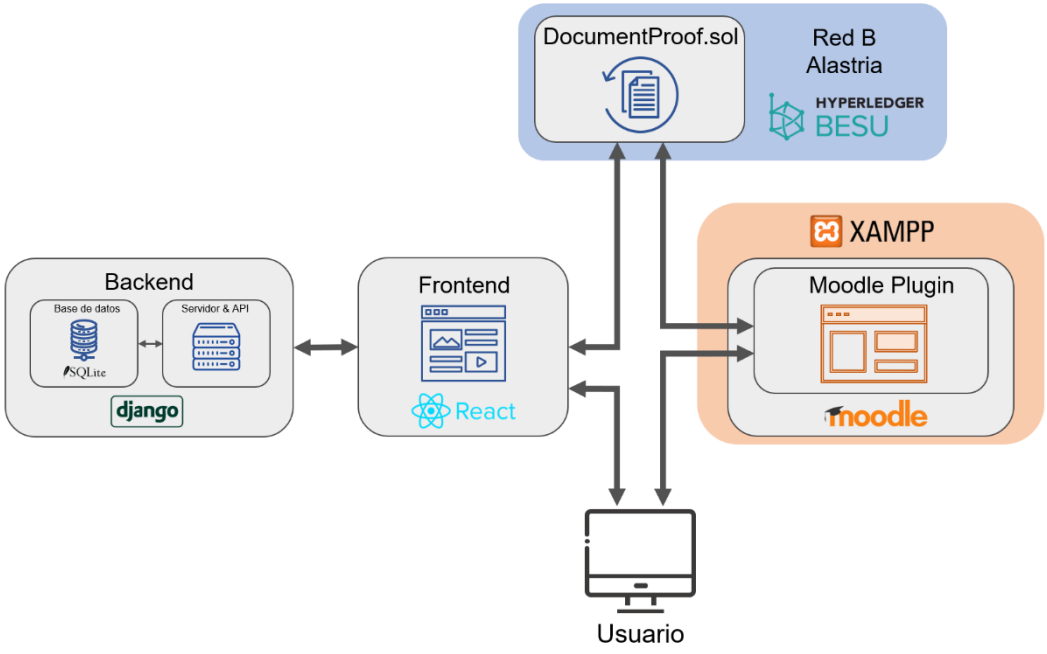


Figura 1. Arquitectura del sistema desarrollado

Resultados

Los resultados de este Trabajo de Fin de Grado han sido mayoritariamente satisfactorios. Se ha logrado desarrollar, implementar y desplegar un smart contract en la Red B de Alastria. Este permite a los usuarios guardar hashes y sus metadatos, editar cierta información, y poder observar la información subida por otras cuentas.

En segundo lugar, se ha desarrollado un frontend con React que permite a los usuarios interactuar con el smart contract en un entorno user-friendly, poniendo el foco en volver más accesibles tecnologías como blockchain. Además, se ha desarrollado un backend capaz de almacenar información para mejorar la experiencia de usuario.

Por último, se ha logrado desarrollar con éxito un plugin de Moodle que permite a los usuarios generar hashes desde los archivos de Moodle, e interactuar con el smart contract.

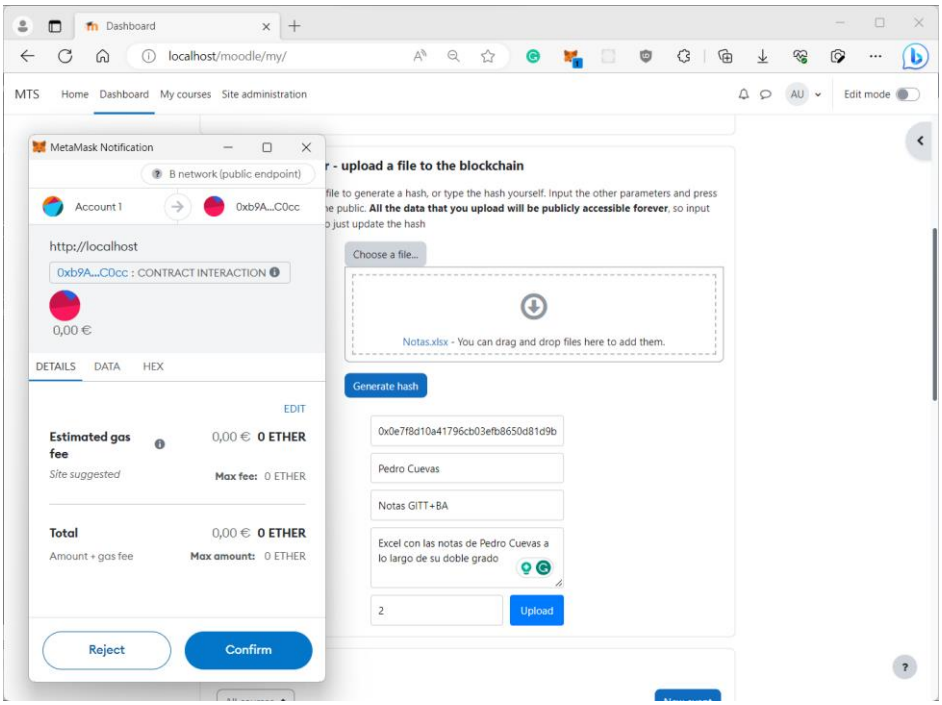


Figura 2. Vista del plugin de Moodle

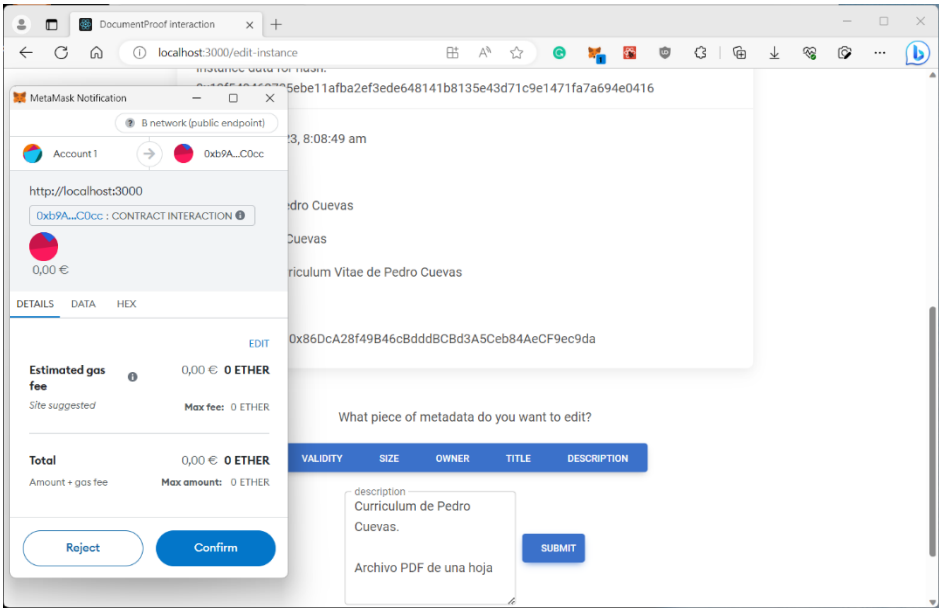


Figura 3. Vista del frontend

Conclusiones

Este proyectó se originó debido a mi deseo de explorar las crecientes oportunidades que ofrece el ecosistema blockchain. En el contexto de una revolución tecnológica es prácticamente imposible predecir cuáles serán las soluciones que triunfen, pero considero que el cumplimiento satisfactorio de los objetivos del trabajo sugiere que el campo de las acreditaciones académicas en redes públicas permisionadas podría ofrecer resultados prometedores en el futuro.

El sistema desarrollado tiene áreas notables de mejora. Estas incluyen:

- Mejora del plugin de Moodle para su despliegue en el Moodle de la Universidad Pontificia Comillas
- Mejora de la interfaz de usuario y expansión del smart contract
- Integración del sistema con otras plataformas educativas

Referencias

[1] Satoshi Nakamoto (2008). «Bitcoin: A Peer-to-Peer Electronic Cash System».

[2] Rodelio Arenas y Proceso Fernandez (2018). «CredenceLedger: A Permissioned Blockchain for Verifiable Academic Credentials».

PROFESSIONAL CREDENTIALS VERIFICATION SYSTEM BASED ON PRIVATE BLOCKCHAIN

Author: Cuevas Olarte, Pedro Roque.

Director: Sanchez Merchante, Luis Francisco.

Collaborating Entity: ICAI – Comillas Pontifical University

PROJECT SUMMARY

The increasing digitization of academic records and the need to guarantee their authenticity has led to the exploration of new technologies that offer security and efficiency. In this context, this Final Bachelor Thesis addresses the challenge of developing an environment that facilitates the accreditation of academic credentials using a public permissioned blockchain network, an emerging technology that offers immutability and transparency in records.

The main focus of the project was the creation and implementation of a smart contract in Alastria's Network B. This network was chosen due to its unique features and its adaptability to handle sensitive data such as academic credentials. The system design also included a frontend and a plugin for Moodle, allowing for a friendly and familiar interface for users. Lastly, a backend was developed using Django and SQLite to extend the system's functionalities and ensure smooth interaction.

On a technical level, the smart contract was initially designed and deployed on an local Hyperledger Besu network, before its final implementation on Network B. Testing on the RESTful API was carried out using Postman , while the frontend was built using MDB and React . The Moodle plugin, meanwhile, was implemented in a local Moodle instance in XAMPP.

The results obtained represent a significant advance in the field of digital academic accreditations. It was possible to build a robust system that allows users to register and validate information, thus facilitating the process of accreditation and authentication of academic documents in digital environments.

Keywords: blockchain, Moodle plugin, academic credentials, Hyperledger Besu, Alastria

Introduction

In today's globalized world, the transparency and authenticity of educational and professional credentials are crucial. Blockchain technology offers an answer to this demand. Providing an immutable and transparent record [1], public permissioned blockchains in particular present a promising solution for authenticating credentials, fusing security with technological innovation [2].

Therefore, our objective is to generate a complete environment that allows users and institutions to prove academic credentials in a private permissioned blockchain network.

Methodology

Project approach

This work seeks to design and create a system that allows improving accessibility, security and transparency to accredit academic credentials.

For this, a smart contract , which will be deployed on Alastria's Network B. This will allow users to store enough information to prove the authenticity of an academic accreditation and a series of metadata to improve the transparency and reliability of the system.

For development, an incremental process will be followed, deploying and testing each element of the architecture independently until the system works cohesively. It will start developing the smart contract . When it works in a local blockchain network, the rest of the elements will be developed in parallel to the final deployment of the contract in Alastria's Network B.

Architecture design

The developed system has an architecture composed of well-separated components:

- Smart contract deployed in network B of Alastria, in charge of registering and allowing consultations of academic accreditations
- Moodle plugin in a local Moodle deployment, which allows the user to interact with the smart contracts
- Frontend: frontend developed in React , which allows the user to interact with the smart contract and with the backend
- Backend: Django backend, which adds functionality to the web to improve the user experience

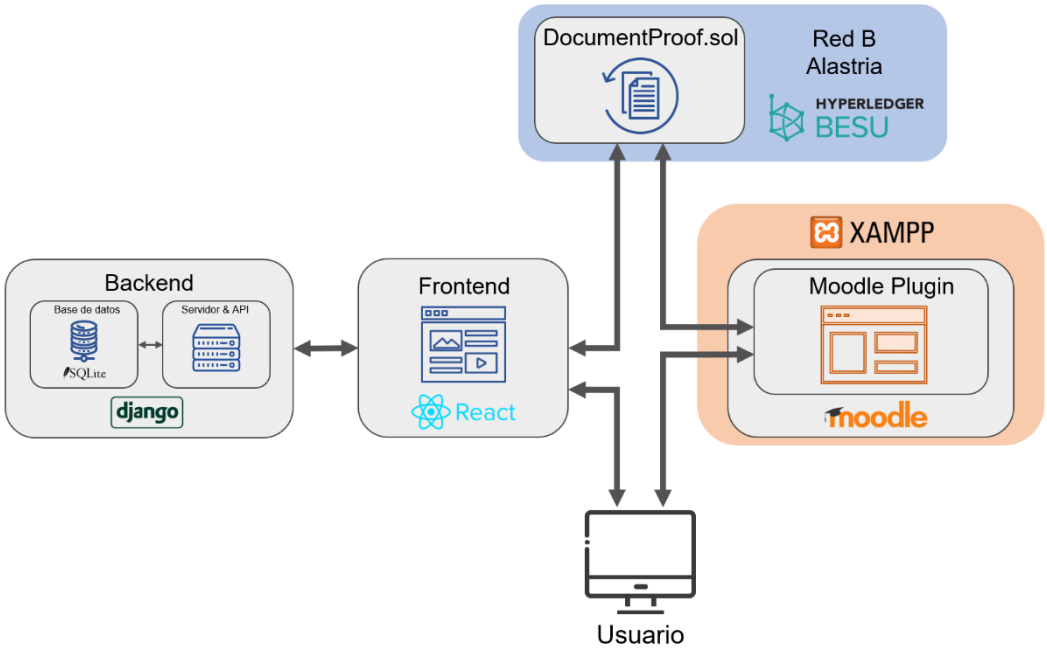


Figure 1. Architecture of the developed system

Results

The results of this Final Degree Project have been mostly satisfactory. It has been possible to develop, implement and deploy a smart contract in Alastria's Network B. This allows users to save hashes and their metadata, edit certain information, and be able to view information uploaded by other accounts.

Secondly, a frontend with React has been developed that allows users to interact with the smart contract in a user-friendly environment, focusing on making technologies such as blockchain more accessible. In addition, a backend capable of storing information has been developed to improve the user experience.

Finally, a Moodle plugin has been successfully developed that allows users to generate hashes from Moodle files, and interact with the smart contract .

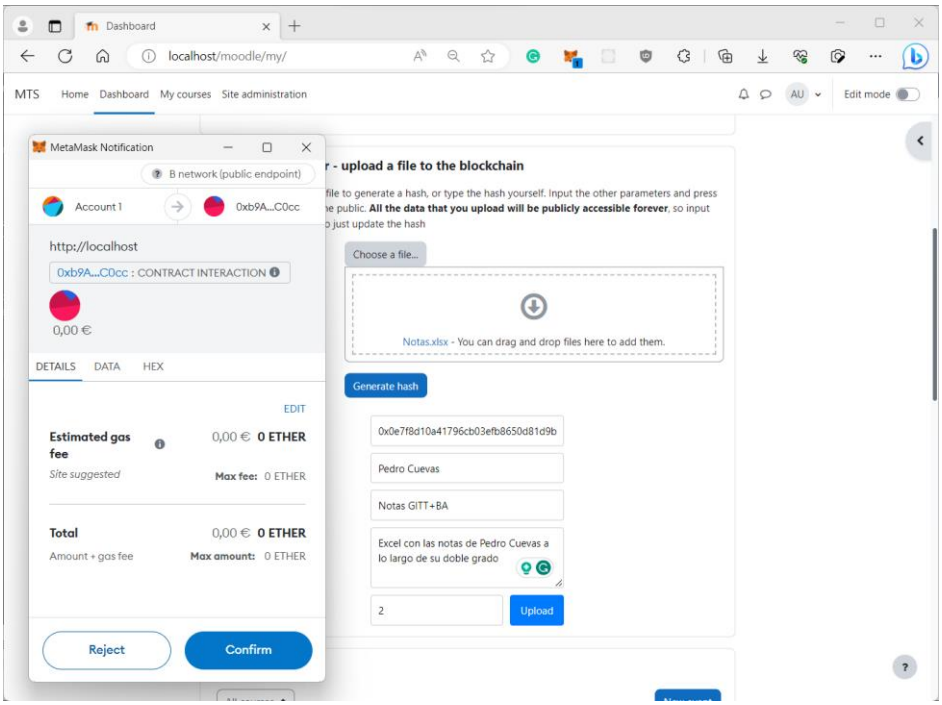


Figura 2. Vista del plugin de Moodle

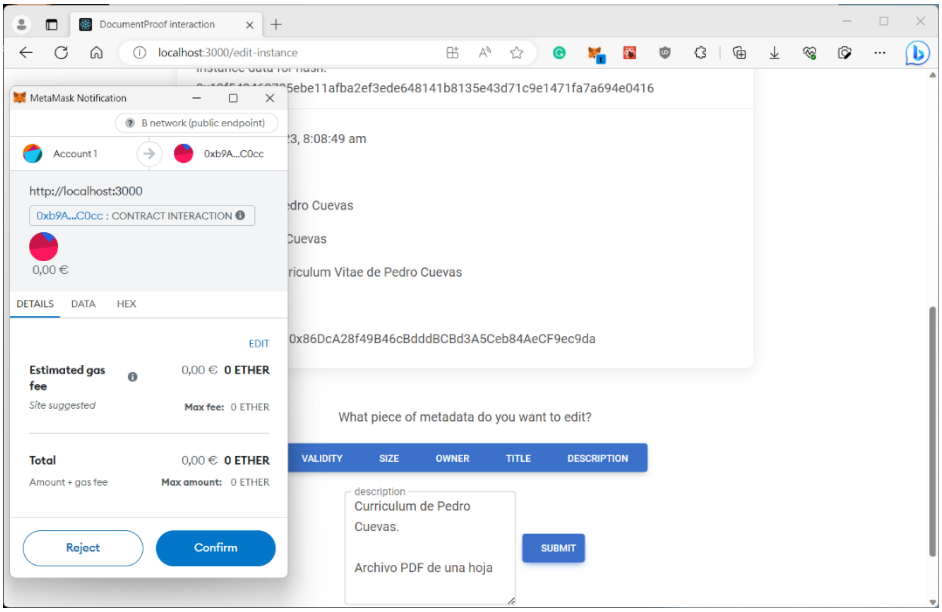


Figura 3. Vista del frontend

Conclusions

This project originated from my desire to explore the growing opportunities offered by the blockchain ecosystem. In the context of a technological revolution, it is practically impossible to predict which solutions will succeed, but I consider that the satisfactory fulfillment of the objectives of the work suggests that the field of academic accreditation in public permissioned networks could offer promising results in the future.

The developed system has notable areas for improvement. These include:

- Improvement of the Moodle plugin for its deployment in the Moodle of the Universidad Pontificia Comillas
- Improvement of the user interface and expansion of the smart contracts
- System integration with other educational platforms

References

[1] Satoshi Nakamoto (2008) . "Bitcoin: A Peer-to-Peer Electronic Cash System".

[2] Rodelio Arenas and Proceso Fernandez (2018). « CredenceLedger : A Permissioned Blockchain for Verifiable Academic Credentials».

Gracias a mi familiar, en especial mis padres, por su paciencia y apoyo. Todos mis logros serán siempre el fruto de su sacrificio y esfuerzo.

Gracias a mis amigos, en especial a Yago, por su apoyo durante este curso.

Grazie Lucrezia, per avermi supportato in questi lunghi mesi.

Finalmetne, mi mayor agradecimiento va a mi director de proyecto, Luis, por su dedicación, entrega y disponibilidad a lo largo del proyecto. En especial, gracias por la paciencia y comprensión ante mis problemas de tiempo debido a mis prácticas.

Índice general

Índice de figuras

Índice de tablas

Índice de código

Capítulo 1

Introducción

En la era contemporánea, la transparencia y la autenticidad en los ámbitos educativo y profesional son más esenciales que nunca. Las instituciones, ya sean educativas o corporativas, enfrentan el desafío constante de discernir la legitimidad de las credenciales presentadas por individuos que buscan oportunidades. Aunque tradicionalmente se ha depositado confianza en la integridad de los currículos y certificados, el mundo globalizado de hoy, caracterizado por una competencia feroz por posiciones prestigiosas, demanda herramientas más sólidas para asegurar una selección basada en logros genuinos. Para ilustrar el problema existente, en 2021, 1 de cada 3 estadounidenses admitió haber mentido en sus currículos [1].

Otro ejemplo anecdótico proviene del panorama político español, donde hace unos años surgieron una serie de escándalos que involucraban a políticos de todo el espectro ideológico proporcionando información falsa o detalles engañosos en sus currículos [2]. Estos incidentes subrayan cómo la falta de veracidad puede erosionar la confianza pública y la credibilidad, planteando la necesidad de enfoques más rigurosos para autenticar las credenciales presentadas en contextos críticos como la política o el mundo profesional.

La necesidad de autenticar logros y credenciales no es un fenómeno reciente. A lo largo de los años, se han desarrollado diversos mecanismos, desde la verificación directa con entidades emisoras hasta la obtención de certificados notariados. A pesar de su utilidad, estos métodos no están exentos de desafíos: pueden ser lentos, propensos a errores humanos y, en ocasiones, resultar costosos. Además, en un mundo cada vez más digital, la necesidad de soluciones que combinen la conveniencia y la seguridad se ha vuelto aún más apremiante.

Es aquí donde la tecnología blockchain surge como una solución potencialmente revolucionaria. Concebida inicialmente por un individuo o grupo bajo el seudónimo

Satoshi Nakamoto en 2008 [3], la blockchain ha trascendido su propósito original como columna vertebral de las criptomonedas. Su capacidad para registrar datos de manera inmutable y transparente ha abierto un abanico de aplicaciones en diversos campos, que van desde la logística hasta la gestión de la cadena de suministro, y desde la atención médica hasta la autenticación de documentos.

Desde la irrupción de Bitcoin, se han establecido múltiples redes blockchain, cada una diseñada con características únicas. Las redes públicas, como Bitcoin y Ethereum, proporcionan una plataforma abierta y descentralizada. Sin embargo, para aplicaciones que manejan datos sensibles, las blockchains privadas han surgido como una opción más adecuada. Ofrecen un mayor grado de control, privacidad y características de seguridad, lo que las hace especialmente aptas para gestionar información como credenciales académicas.

En este contexto, este trabajo se adentra en la exploración de una solución basada en una blockchain privada. El objetivo es desarrollar un sistema que permita a entidades educativas y laborales certificar y validar logros y experiencias académicas con una confiabilidad sin precedentes. La meta final es establecer un estándar unificado que, al aprovechar la tecnología blockchain, pueda garantizar la autenticidad y legitimidad del recorrido de un individuo. A través de la integración de la seguridad inherente y la transparencia de la blockchain, esta investigación se esfuerza por contribuir a un futuro donde la confianza en las credenciales se fortalezca mediante la innovación tecnológica.

Capítulo 2

Descripción de las tecnologías

En este capítulo se ofrecerá una breve descripción de las principales tecnologías a las que se hace referencia en este Trabajo de Fin de Grado, así como el papel que han tenido, si fuera relevante.

2.1. Blockchain

La tecnología blockchain ha surgido como una solución transformadora en la gestión y verificación de datos. Sus orígenes pueden rastrearse hasta la introducción de Bitcoin en 2009 por un paper escrito bajo el seudónimo de Satoshi Nakamoto [3]. Fundamentalmente, el blockchain opera como un sistema de registro descentralizado y criptográfico que ha revolucionado los métodos tradicionales de almacenamiento de datos y autenticación.

Funcionando como una cadena secuencial de "bloques" interconectados, cada bloque contiene datos encriptados y un enlace criptográfico al bloque precedente. El principio clave subyacente en el blockchain es el consenso distribuido, mediante el cual una red de participantes valida colectivamente las transacciones a través de protocolos computacionales. Esta validación, a menudo implementada mediante mecanismos como la prueba de trabajo o la prueba de participación, asegura la integridad e inmutabilidad de los datos almacenados. A medida que se añaden nuevos bloques, la cadena completa se vuelve cada vez más segura y resistente a alteraciones no autorizadas [3].

Algunos de los mecanismos de consenso más comunes [4] se enumeran a continuación:

- **Proof of Work (PoW):** el primer minero en resolver correctamente una ecuación puede añadir el siguiente bloque y recibir un premio. Aunque es uno de los métodos que genera una mayor fiabilidad, es lento, ineficiente y consume mucha energía.
- **Proof of Stake (PoS):** los nodos validadores deben comprometer unidades monetarias. En caso de fraude, las perderán. Pueden recibir recompensas en caso contrario. Resuelve la mayoría de los problemas de PoW, pero a costa de que los validadores con gran cantidad de monedas puedan tener demasiada influencia.
- **Proof of Activity:** una combinación de PoW y PoS.
- **Proof of Authority (PoA):** existen una serie de nodos permisionados, que pueden autorizar transacciones mediante un mecanismo de consenso. Es rápido y escalable, pero requiere que ya exista confianza previa en los nodos validadores. Por lo tanto, en ciertos contextos una base de datos distribuida tradicional podría ser preferible [5].

2.1.1. Ethereum

Ethereum tuvo su origen en un white paper escrito por Vitalik Buterin en 2013, en el cual esbozaba una visión para una plataforma que expanda las innovaciones introducidas por Bitcoin [6]. Esta plataforma tenía como objetivo ejecutar contratos inteligentes y aplicaciones descentralizadas (conocidas como DApps en inglés).

Lanzado en 2015, la red Ethereum opera como una plataforma de blockchain descentralizada utilizando Ether (ETH) como su criptomoneda nativa. Es de código abierto, y soporta contratos inteligentes y DApps.

2.1.2. Tipos de redes blockchain

Originalmente, las redes blockchain consistían en plataformas puramente públicas. Esto implica que cualquier miembro de la red podía actuar como parte del proceso de consensos, realizar transacciones y leer el historial de estas. Sin embargo, con el tiempo han surgido distintos tipos de redes blockchain, que asignan roles y permisos a sus miembros [5]:

- **Redes públicas sin permisos:** son redes blockchain como Bitcoin. Cualquier miembro de la red puede participar, enviar transacciones y leer de la

cadena de bloques. Es la más tradicional y está presente en la mayoría de criptomonedas.

- **Redes privadas sin permisos:** funcionan igual que las redes públicas sin permisos, con la excepción de que el acceso a la red está restringido a un grupo de miembros.
- **Redes públicas permisionadas:** los permisos de escritura están restringidos a un grupo de entidades, pero los permisos de lectura están abiertos para cualquiera con acceso a internet. Es de interés cuando tenemos una red para las que las que existe cierto grado de confianza en las entidades permisionadas.
- **Redes privadas permisionadas:** en este caso, tanto la lectura como la escritura están restringidas a un grupo concreto de miembros, generando una estructura jerárquica basada en roles. El uso de blockchain se vuelve cuestionable en este contexto, pues sus principales ventajas de blockchain podrían tener una aplicación demasiado limitada, lo que haría más adecuado el uso de bases de datos tradicionales.

Como se puede ver, la variedad de opciones ofrece diferentes ventajas además de existir numerosos compromisos entre diferentes variables. A continuación, se detallan los más habituales.

2.1.3. Eje transparencia-privacidad

La transparencia garantiza que los usuarios puedan verificar la veracidad de un certificado, así como la confianza en la red. Sin embargo, la privacidad puede ser necesaria en algunos contextos. En el ámbito educativo, puede ser esencial para no revelar información protegida de los aplicantes. Por lo tanto, es necesario encontrar un equilibrio entre las dos características. Una posible solución sería que en la red blockchain se almacenen identificadores del documento a acreditar. Mientras tanto, sería el propio aplicante el que cede el documento a la entidad interesada, o esta última podría obtener el documento de una base de datos tradicional [5]. De esta manera la veracidad del documento podría ser comprobable, al tiempo que se protege la información personal relacionada con el certificado.

El candidato ideal es una clave hash generada por una función criptográfica hash unidireccional. Estas funciones se caracterizan por producir, a partir de un input de longitud arbitraria, un output de siempre la misma longitud. Además, aun sabiendo el código hash y la función usada, es computacionalmente inasumible averiguar el input. Una de las características de estas claves es que cualquier pequeño cambio en

el input genera grandes cambios en el output [7]. Aprovechando esta propiedad, si almacenáramos de manera pública claves hash, se podría garantizar la protección de la información del certificado. Cualquier persona con acceso al título podría calcular su clave hash y verificar su existencia en la blockchain, así como qué entidad emitió el certificado (en el caso de tratarse de una blockchain pública) [5]. Una red blockchain permisionada añadiría una capa adicional de seguridad, al garantizar que solo las entidades autorizadas puedan emitir certificados (evitando el problema de las fábricas de títulos falsos).

2.1.4. Eje rendimiento-fiabilidad

Una red blockchain es inherentemente más lenta que una base de datos tradicional. Esto ocurre porque existe una latencia asociada a la sincronía entre nodos, así como al mecanismo de consenso empleado [5].

2.1.5. Smart contract

En el ámbito de Ethereum, un contrato inteligente es esencialmente un contrate que puede ejecutarse por sí mismo haciendo que se cumplan condiciones o acuerdos predefinidos sin necesidad de intervención manual. Estos contratos encuentran su alojan en la Máquina Virtual Ethereum (EVM), que actúa como un entorno en el que los contratos inteligentes se ejecutan con integridad y son verificados por la red.

Cuando se inicia un contrato inteligente en Ethereum, inicialmente se escribe con instrucciones y condiciones utilizando lenguajes como Solidity. Una vez desplegado en la red Ethereum, recibe una dirección. Las personas que participan en un contrato activan su ejecución enviando transacciones que activan sus funciones. Estas transacciones se comparten en toda la red Ethereum. Comprobadas por los nodos mediante mecanismos de consenso. El EVM se encarga del procesamiento. Ejecutar el código dentro del contrato, lo que puede implicar cálculos cambios en los datos almacenados o interacciones con los contratos. En cuanto se produce esta ejecución los resultados pasan a formar parte de un registro en la blockchain garantizando la transparencia, inmutabilidad y trazabilidad durante todo el proceso. Los contratos inteligentes no permiten automatizar la confianza en los acuerdos. También allanan el camino para aplicaciones descentralizadas, con amplios usos en el mundo real.

2.1.6. Hash

Un hash es una función matemática que convierte una entrada en una cadena de caracteres de longitud fija, que normalmente parece ser aleatoria. Esta salida, conocida como hash, es única para cada entrada única; incluso un pequeño cambio en la entrada producirá una salida completamente diferente. Una característica crucial de las funciones hash es que son unidireccionales, lo que significa que, a partir de la salida, es computacionalmente inviable regenerar la entrada original [8].

En el contexto de las credenciales académicas, el hash puede ser utilizado como una herramienta de verificación. Por ejemplo, al emitir un diploma o certificado, una institución puede calcular el hash del documento y almacenarlo en un lugar seguro o en una base de datos distribuida como blockchain. Cuando alguien quiera verificar la autenticidad de un documento en el futuro, simplemente puede calcular el hash del documento presentado y compararlo con el hash almacenado originalmente. Si coinciden, el documento es auténtico; de lo contrario, ha sido alterado.

En mi investigación, he decidido usar la técnica de hashing para almacenar credenciales académicas en una plataforma descentralizada. Al guardar solo el hash del documento y no el documento en sí, se garantiza la privacidad de la información, además de obtener una forma mucho más eficiente de guardar información.

2.1.7. Hyperledger Besu

Hyperledger Besu es un cliente de Ethereum de código abierto desarrollado bajo la licencia Apache 2.0 y escrito en Java. Está diseñado para ser compatible con redes públicas y privadas, tanto permisionadas como no permisionadas, así como con redes de prueba como Sepolia y Görli. Además, incluye varios algoritmos de consenso, como Proof of Stake, Proof of Work, Proof of Authority e IBFT 2.0.

El cliente de Ethereum permite el desarrollo, despliegue y uso operativo de contratos inteligentes y aplicaciones descentralizadas, utilizando herramientas como Truffle, Remix o web3j. Hyperledger Besu es parte de la fundación Hyperledger, que es un proyecto colaborativo de código abierto que tiene como objetivo promover las tecnologías de contabilidad distribuida entre diversas industrias. La fundación busca ofrecer una solución de cliente de Ethereum que sea flexible, segura, escalable y compatible con los estándares de la industria. Asimismo, también facilita la interoperabilidad entre redes de Ethereum y otras plataformas de Hyperledger, como Fabric y Sawtooth.

Hyperledger Besu ha sido la red blockchain utilizada en este trabajo para almacenar información, nediante un smart contract desplegado en la misma.

2.1.8. Metamask

Metamask es una extensión de navegadores que actúa como cartera para poder interactuar y firmar transacciones con redes blockchain. Aunque fue construida para Ethereum, es compataible con otras redes blockchain, como Hyperledger Besu.

Algunas de sus funcionalidades principales son las de administrar criptomonedas, acceder a dApps, y firmar transacciones.

2.1.9. Remix IDE

Remix IDE es un entorno de desarrollo online que permite programar, compilar, testear e implementar smart contracts con Ethereum y otras redes blockchain compatibles (como Hyperledger Besu).

Algunas de sus funcionalidades más destacadas son las de incluir un compilador de Solidity, debugging y herramientas para interactuar con contratos ya desplegados.

2.1.10. Truffle

Truffle es un marco de desarrollo de código abierto diseñado para hacer más accesible y eficiente la creación y gestión de contratos inteligentes en plataformas blockchain. Específicamente enfocado en la red Ethereum y sus variantes, Truffle proporciona una suite de herramientas que abarca desde la escritura y prueba de contratos inteligentes hasta la migración y despliegue de aplicaciones dApps. Con una combinación de compilación automatizada, gestión de dependencias y pruebas integradas, Truffle simplifica considerablemente el proceso de desarrollo para los desarrolladores de contratos inteligentes, evitando problemas derivados de detalles técnicos [9].

En este Trabajo de Fin de Grado, Truffle se ha usado principalmente para desarrollar una red Hyperledger Besu local para desplegar y probar el smart contract.

2.2. Backend

El backend se trata de la parte del servidor de una aplicación de software. Se encarga de tareas como el procesamiento de datos, el almacenamiento y la comunicación entre la interfaz de usuario (frontend) y la base de datos. En esencia, el backend gestiona las solicitudes de los usuarios, procesa datos y genera respuestas que luego se muestran a los usuarios a través del frontend.

En términos de funcionalidad, el backend opera recibiendo solicitudes desde el frontend (típicamente a traves de una API), gestionándolas y generando respuestas. Se encarga de la lógica relacionada con la interacción con bases de datos, servicios externos y otros recursos según sea necesario para satisfacer las necesidades del sistema.

A continuación, detallaré las tecnologías utilizadas en el context del backend de este trabajo de fin de grado.

2.2.1. Python

Python es un lenguaje de programación multipropósito que se utiliza con frecuencia para desarrollar la parte del backend de aplicaciones de software. Es conocido por su facilidad de uso, versatilidad y ecosistema de bibliotecas y frameworks, que facilitan el proceso de desarrollo y permiten generar backends robustos.

La razón por la que se eligió Python como lenguaje de desarrollo es debido a su flexibilidad, que permite la creación de prototipos y ciclos de desarrollo eficientes. Su naturaleza orientada a objetos y su compatibilidad con frameworks como Django y Flask permiten a los desarrolladores construir arquitecturas backend escalables y mantenibles. Además, Python funciona sin problemas en diferentes sistemas operativos y sobresale en tareas como la manipulación de datos, la integración de API y la gestión de bases de datos. Esto lo convierte en una excelente elección para construir funcionalidades desde cero.

2.2.2. DJango

Django es un framework de desarrollo web basado en Python. Algunas de sus características principales incluyen los principios de diseño definidos y la estructuración eficiente de aplicaciones web, que a su vez permiten escalabilidad, mantenibilidad y gestión eficaz de datos. Algunas características clave que hacen idóneo a Django son [10]:

- Arquitectura MVC: Django sigue el patrón Model-View-Controller (MVC) asegurando una separación de responsabilidades y promoviendo un código modular fácil de implementar.
- Integración ORM: con su sistema Object Relational Mapping (ORM) los desarrolladores pueden interactuar con la base de datos utilizando clases de Python en lugar de escribir consultas SQL directas. Esto no mejora la eficiencia del código y simplifica las operaciones de base de datos.
- Interfaz de administración: Django proporciona una interfaz que genera automáticamente una interfaz de usuario amigable para la gestión de datos en la aplicación.
- Enrutamiento de URL: el enrutador de URL en Django simplifica la configuración de patrones de URL, facilitando la creación de endpoints.
- Django viene con medidas de seguridad que mejoran la robustez de la aplicación

2.2.3. SQLite

SQLittle es un motor de base de datos SQL que permite el almacenamiento de datos de una manera ligera y eficiente.

Algunas de sus características clave son [11]:

- Arquitectura sin servidor: SQLittle simplifica el proceso de despliegue y mantenimiento encapsulando la base de datos dentro de un archivo eliminando la necesidad de gestión del servidor.
- Cumplimiento ACID: la base de datos se adhiere estrictamente a los principios de Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID) garantizando que las transacciones sean seguras y fiables.
- Naturaleza autónoma: las bases de datos SQLittle son archivos autónomos, lo que significa que pueden transportarse y desplegarse fácilmente sin dependencias. Esto las hace adecuadas para entornos.
- Compatibilidad SQL: SQLittle sigue el lenguaje SQL permitiendo a los desarrolladores familiarizados con la consulta SQL interactuar sin problemas con ella.
- Integración con Django: SQLittle se integra sin problemas con el framework Django permitiendo la gestión de datos a través de las capacidades de Object Relational Mapping (ORM) de Djangos.

- A pesar de ser ligero, SQLittle ofrece un rendimiento de consulta que es crucial para los proyectos que manejan grandes cantidades de datos.

2.2.4. Postman

Postman es una herramienta de interacción con APIs. Facilita la creación, testeo y documentación de APIs, mediante una interfaz fácil de usar.

En concreto, su funcionalidad más útil para este Trabajo de Fin de Grado ha sido la de poder configurar y realizar llamadas a una API con facilidad.

2.3. Frontend

El frontend, a menudo denominado client-side, es el aspecto de una aplicación de software orientado a usuario. Esto incluye los elementos interactivos con los que los usuarios interactúan directamente, como interfaces de usuario, diseños y experiencias de usuario en general. El desarrollo frontend implica la creación de estos componentes utilizando tecnologías como HTML, CSS y JavaScript (o React en nuestro caso) para garantizar una interacción efectiva con el usuario.

2.3.1. React

React es una biblioteca JavaScript creada por Facebook para generar interfaces de usuario. Facilita el desarrollo de interfaces de usuario mediante el uso de componentes que gestionan tanto la lógica como la presentación [12].

Algunas de sus características destacables son:

- Programación declarativa: React sigue un modelo de programación para gestionar las manipulaciones del Modelo de Objetos del Documento (DOM) haciendo que el código sea más predecible.
- Arquitectura basada en componentes: Los componentes en React permiten el desarrollo de UI haciendo más fácil la reutilización de código y la creación de composiciones.
- DOM virtual: el DOM virtual de React optimiza la representación reduciendo el número de actualizaciones realizadas en el DOM, lo que mejora el rendimiento.

- Flujo de datos unidireccional: React asegura la propagación de datos simplificando la gestión del estado y facilitando la depuración.
- Sintaxis JSX: JSX tiene similitudes con HTML. Simplifica la integración de la interfaz de usuario con la lógica.

En resumen, el enfoque de React de ser declarativo, su estructura basada en componentes, la optimización a través de DOM y la integración con JSX permiten a los desarrolladores crear de manera eficiente interfaces de usuario fáciles de mantener.

2.3.2. Bootstrap

Bootstrap es un framework frontend muy popular en desarrollo web. Es una herramienta de código abierto que ofrece una gran variedad de componentes, estilos y utilidades de diseño; simplificando el proceso de desarrollo frontend. Con Bootstrap se facilita la creación de aplicaciones web visualmente atractivas.

2.3.3. Material Design for Bootstrap

MDB (Material Design for Bootstrap) es otro kit de interfaz de usuario integrado con Bootstrap. Sigue los principios de Material Design de Google. Ofrece componentes y plantillas con estilo, que combinados con React MDB permiten a los desarrolladores construir aplicaciones de usuario manteniendo elementos de diseño responsivos y estéticos.

2.4. Moodle

Moodle, es un LMS (Learning Management System - Sistema de Gestión de Aprendizaje) de código abierto, siendo uno de los LMS más populares actualmente. Se utiliza en gran número de instituciones educativas para crear entornos de aprendizaje online.

Cuenta con un ecosistema de plugins desarrollados por la comunidad, lo que permite que existan una gran cantidad de recursos disponibles para ayudar en el desarrollo de plugins.

2.4.1. XAMPP

XAMPP es un paquete de software libre. Combina un sistema de gestión de bases de datos (MariaDB), con un servidor Apache e intérpretes de lenguajes de programación como PHP. Su utilidad en este Trabajo de Fin de Grado radica en que es una herramienta idónea para ejecutar Moodle [13].

2.5. Oracle VM Virtualbox

Oracle VM VirtualBox es un software de virtualización que permite crear y alojar máquinas virtuales. Esto significar que permite simular sistemas operativos como Windows o múltiples distribuciones de Linux. Como consecuencia, ayuda a probar la compatibilidad de un software realizando experimentos virtuales y creando entornos de desarrollo aislados.

2.6. Visual Studio Code

Visual Studio Code, también conocido como VS Code, es una herramienta de edición de código creada por Microsoft. Proporciona una gran cantidad de funciones y extensiones, lo que lo hace muy versátil para programar con varios lenguajes y frameworks de programación.

Es la herramienta principal utilizada para escritura de código en este Proyecto de Fin de Grado.

Capítulo 3

Estado del Arte

3.1. Herramientas de verificación tradicionales

Desde siempre, la autenticidad de los documentos académicos ha sido una piedra angular para preservar la integridad tanto en contextos educativos como profesionales. Con el paso de los años, se han adoptado diversos mecanismos para este fin. Ahondaremos en las prácticas más reconocidas en esta sección.

3.1.1. La solidez de las Notarías

Las notarías, respaldadas por profesionales capacitados conocidos como notarios públicos, han sido un pilar fundamental en la autenticación de documentos durante generaciones. La firma y el sello de un notario certifican la autenticidad de un documento, proporcionándole una fe pública con plena validez jurídica [14]. Aunque este enfoque es confiable y robusto, puede carecer de agilidad, y en ocasiones, supone un costo elevado, especialmente en contextos internacionales donde se requiere una apostilla. Por ejemplo, en Madrid, la tarifa de una apostilla varía entre 18 y 37€ [15]. No es exorbitante, pero puede desalentar a quienes no estén obligados a realizar tal trámite.

3.1.2. El auge de las Firmas Digitales

El avance tecnológico nos ha legado herramientas revolucionarias, como la firma digital. Esta herramienta usa criptografía para vincular la identidad del firmante con el contenido del documento. Cualquier intento de alteración posterior invalidará la

firma, evidenciando la manipulación [16]. A pesar de su velocidad y eficiencia, es vital confiar en el proveedor del servicio de firma digital, ya que aún es susceptible a amenazas cibernéticas [17].

3.1.3. Validación directa a través de la Institución

Es común solicitar directamente a la entidad educativa una validación oficial que avale un documento. Estos certificados, sellados y firmados, confirman la legitimidad del documento en cuestión. Si bien este método es confiable, puede tornarse tedioso, sobre todo al tratar con múltiples documentos o instituciones distantes.

Con esto en mente, se destaca que, aunque las técnicas convencionales son efectivas, presentan desafíos en términos de eficiencia y coste. Surge, entonces, una inevitable interrogante: ¿podrían las tecnologías emergentes, como blockchain, superar estos obstáculos?

Diversas iniciativas buscan abordar esta cuestión, cada una con sus propios enfoques y consideraciones:

- **Blockcerts:** se trata de una propuesta del Massachusetts Institute of Technology. En ella, se almacenan códigos (como en el caso de claves hash explicado anteriormente) en redes blockchain de acceso público (como Ethereum o Bitcoin). Por lo tanto, sigue existiendo el problema anteriormente mencionado de fábricas de títulos [5]. En la actualidad, los alumnos del MIT tienen la posibilidad de recibir un certificado de su título en Blockcerts al graduarse [18].
- **Disciplina:** es similar a Blockcerts, con la particularidad de que se centra en crear un marketplace para reclutadores [5].
- **Sony Global Education:** iniciativa lanzada por Sony Global Education bajo una alianza con el gobierno de Japón y una red blockchain basada en la nube de IBM. El rol del gobierno japonés garantiza que las instituciones participantes estén acreditadas. Sin embargo, el uso de utilizar una red blockchain que funciona en la nube de una única entidad es contradictorio con el propósito de blockchain, al obligarnos a confiar en dicha entidad [5].
- **BCDiploma:** Se trata de una iniciativa francesa. Utiliza Ethereum para guardar claves hash de los diplomas que acredita. Sufre de un problema similar a Sony Global Education, ya que cuenta con un servicio intermediario llamado EvidenZ, en el que las instituciones académicas confían. Esto tiene el potencial de disminuir las ventajas de usar la tecnología blockchain, ya que es necesario confiar en un tercero para utilizar el servicio [5].

- **Edgecoin:** se centra en construir un modelo Business-To-Business, ya que conecta instituciones académicas, otro tipo de instituciones educativas y plataformas de talento. Su funcionamiento se basa en el almacenaje de hashes, de manera análoga a las anteriores iniciativas. Además, cuenta con la particularidad de que usa IPFS para guardar los diplomas originales. IPFS es una tecnología que permite almacenar archivos de manera descentralizada en internet. Si bien está información podría estar encriptada, sigue existiendo el riesgo de filtraciones de datos privados [5].
- **Open Source University:** es una iniciativa similar a Edgecoin. Los diplomas académicos se guardan de manera descentralizada en una aplicación descentralizada llamada OSU [5].
- **CredenceLedger:** se basa en una red blockchain permisionada. Las razones para hacerlos son que esta permite métodos de consenso más eficientes y mayor espacio [19]. Sin embargo, el uso de una red permisionada supone sacrificios en las características de transparencia y confiabilidad, especialmente sensibles en el contexto de las acreditaciones académicas [5].

El panorama español destaca por la presencia de Alastria, una red compuesta por más de 500 socios, incluyendo empresas, instituciones académicas y entidades públicas [20].

La más activa de sus redes es la Red-B, impulsada por la tecnología Hyperledger BESU, un cliente Ethereum de código libre. Emplea el mecanismo de consenso IBFT 2.0, fundamentado en Proof of Authority (PoA) [21]. Es, por lo tanto, una red pública pero con acceso restringido.

Dentro de este marco es de destacar el caso de Open Digital Registry, desarrollado por Open Canarias SL, que permite registrar hashes en la blockchain de Alastria y verificar su autenticidad a través de su plataforma web [22].

El interés académico de enfocar mi Trabajo de Fin de Grado en las particularidades de la Red B de Alastria radica en sus características únicas. Siendo una red pública permisionada, combina la transparencia y apertura de las redes públicas con la seguridad y control de las redes privadas. Además, el respaldo de múltiples entidades en España y la adaptabilidad de Hyperledger BESU sugieren que esta red puede ofrecer soluciones avanzadas y personalizadas para la autenticación de documentos, posicionándola como una alternativa potencial a las metodologías tradicionales.

Capítulo 4

Definición del Trabajo

4.1. Motivación

La propuesta de este trabajo está motivada por los siguientes hechos:

- Los distintos tipos de tecnologías blockchain que se pueden usar implican sacrificios en los ejes transparencia-privacidad y rendimiento-fiabilidad. Todavía no se han explorado todas las posibilidades que ofrecen los distintos compromisos en estos ejes.
- De entre las opciones comentadas, las redes públicas permisionadas ofrecen un equilibrio adecuado para el contexto del presente trabajo, ya que:
    - Una entidad ha de estar autorizada para poder añadir bloques nuevos a la cadena, lo que exige un cierto grado de confianza, pero aumenta el rendimiento.
    - Son más escalables, eficientes y más rápidas que las redes sin permisionar.

Este trabajo de fin de grado se centra en la construcción de una herramienta que aproveche las características de la red B de Alastria (pública permisionada) para poder acreditar documentos de la vida profesional y académica de un individuo. También se pretende construir un plugin de Moodle que permita integrar las entregas que un estudiante realiza durante su formación (trabajos, presentaciones, TFG, TFM, tesis doctoral, ...) en la cadena de bloques, certificando de esta manera su trayectoria académica. Esta propuesta supone una solución integradora e innovadora en el contexto español, inexistente en el momento de la redacción de este trabajo.

4.2. Objetivos

Tal y como se ha desarrollado a lo largo de este Trabajo de Fin de Grado, nuestro objetivo principal es el de desarrollar un sistema completa que permita dar acceso al mayor número de personas posibles a blockchain, con tal de que puedan guardar y verificar credenciales académicas.

Para lograrlo, los objetivos definidos en este trabajo de fin de grado se dividen en principales y secundarios.

4.2.1. Objetivos principales

- Desarrollar un smart contract que permita certificar credenciales académicas, o documentos relacionados con la trayectoria de un estudiante universitario. Este smart contract debería permitir almacenar un identificador tipo hash del archivo y metadatos relevantes.
- Crear un frontend básico que permita al usuario interactuar con el smart contract

4.2.2. Objetivos secundarios

- Desarrollo de un plugin para Moodle que permita integrar las entregas académicas dentro de la cadena de bloques
- Desplegar smart contract en la Red B Alastria
- Diseñar un backend básico para complementar el frontend. Este backend deberá comunicarse con el frontend mediante una REST API, para que eventualmente terceros autorizados puedan hacer uso de dicha API
- Integrar esta aplicación en el Moodle de la Universidad Pontificia Comillas
- Deplegar y testear el funcionamiento de todos los elementos del sistema

La consecución de estos objetivos está supeditada a ir resolviendo todos los problemas tecnológicos o administrativos que puedan surgir durante este trabajo de fin de grado.

4.3. Metodología de trabajo

El trabajo propuesto se basará en una aplicación que implemente smart contracts en la red B de Alastria, basada en HyperLedger Besu.

El desarrollo de la aplicación se realizará sobre un despliegue local de una red basada en HyperLedger Besu, que permitirá probar los prototipos antes de su despliegue definitivo en la red B de Alastria.

A medida que se vaya avanzando en el desarrollo de la aplicación de certificación, se iniciará el desarrollo del frontend, el backend, la API, y el plugin para Moodle.

El frontend, backend y la API se desarrollarán y testearán en archivos locales, para luego desplegarlos.

Para el desarrollo del plugin de Moodle, se desplegará un servidor local para alojar un sitio web Moodle donde pueda ser probado.

La redacción del TFG se realizará de manera paralela al resto de tareas del proyecto.

El seguimiento del trabajo se basará en la metodología SCRUM, ya que se mantendrá una reunión semanal para marcar el cumplimiento de tareas, asignar nuevas tareas semanas y comentar el desarrollo del proyecto.

4.4. Planificación

El cronograma representado en la figura 4.1 resume las estimaciones actuales del desarrollo del proyecto. Por lo tanto, está supeditado a los cambios generados por las posibles circunstancias que surjan a lo largo del mismo.

A continuación, adjuntaré una pequeña descripción de la metodología a seguir para cumplir ciertas tareas:

- **Aprendizaje de Solidity**
  - Documentación sobre el lenguaje
  - Puesta en marcha de entorno de desarrollo (Remix IDE)
  - Desarrollo de smart contracts de prueba
- **Despliegue de red de desarrollo local**
  - Creación de máquina virtual en Linux

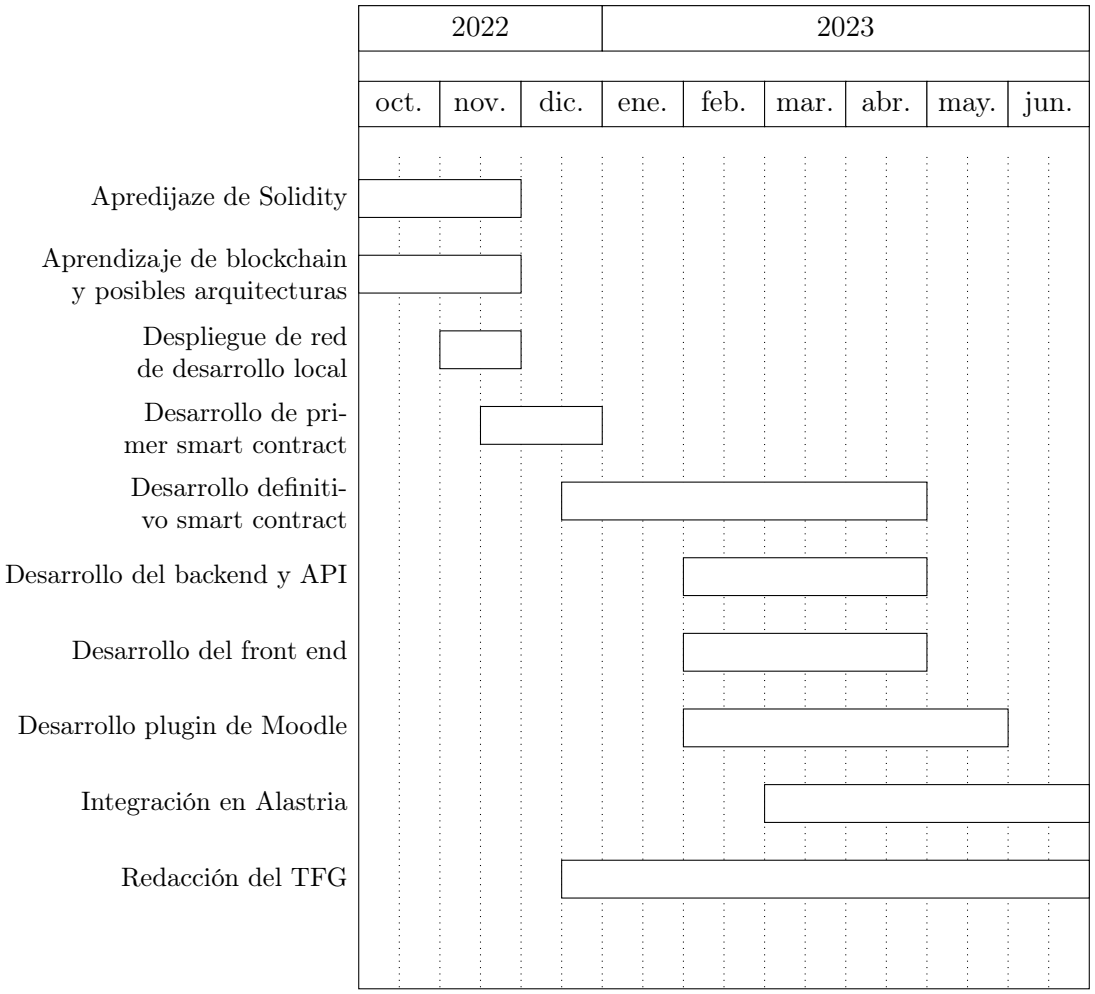


Figura 4.1: Planificación del trabajo

  - Puesta en marcha de nodo local de Hyperledger Besu
  - Testeo de API para interactuar con el nodo local
- **Despliegue de primer smart contract**
  - Desarrollo de smart contract
  - Despliegue en red local
  - Testeo
- **Desarrollo de versión definitiva de smart contract**
  - Diseño de smart contract definitivo
  - Desarrollo de smart contract
  - Despliegue en red local
  - Testeo
- **Desarrollo de backend y API**
  - Documentación y aprendizaje de Django
  - Desarrollo en Windows local
  - Testeo de la API
- **Desarrollo de frontend**
  - Documentación y aprendizaje de React
  - Desarrollo en Windows local
  - Testeo
- **Desarrollo del plugin de Moodle**
  - Documentación y aprendizaje de Moodle y PHP
  - Despliegue de Moodle local con XAMPP
  - Desarrollo del plugin
- **Integración en Alastria**
  - Pedir análisis del smart contract a personal de Alastria
  - Incorporar feedback
  - Desplegar contrato en Red B de Alastria

- Testeo con Remix IDE
- Testeo de Plugin de Moodle y frontend

4.5. Planificación económica y recursos utilizados

En este apartado se realizará un análisis económico del desarrollo del trabajo. Esto incluye el coste del desarrollo y una estimación del coste de mantenimiento del sistema. El sistema desarrollado no tiene ánimo de lucro, por lo que no se hará un análisis de ingresos.

4.5.1. Recursos materiales

El único recurso material utilizado para el dearrollo del Trabajo de Fin de Grado ha sido el ordenador portátil descrito en la tabla 4.1

Concepto	Especificación
Nombre	Lenovo Yoga 920-13IKB
Procesador	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
Tarjeta gráfica	Intel(R) UHD Graphics 620
Memoria RAM	8.00 GB DDR4 (7.84 GB disponibles)
Almacenamiento	500GB SAMSUNG MZVLW512HMJP-000L2
Pantalla	13.9" (1920x1080)
Sistema operativo	Windows 11 Home
Precio actual estimado	600 €

Tabla 4.1: Especificaciones de ordenador portátil

Como es un portátil que ya tenía a mi disposición antes de realizar el presente Trabajo de Fin de Grado, es necesario estimar sus horas de uso y el coste de las mismas.

Según la agencia tributaria, un periodo razonable par depreciar un portátil podrían ser 4 años [23]. Teniendo en cuenta que la duración de este trabajo de fin de grado ha sido de aproximadamente 3h/día a lo largo de 10 meses, las horas totales serían: $10 * 30 * 3 = 900h$

Como 4 años de 8h de trabajo diaria en días laborales son: $250 * 8 * 4 = 80000h$, el coste en depreciación del portátil sería de $900/8000 = 11{,}25\,\%$, que son 76,50 €.

4.5.2. Recursos humanos

El salario medio bruto de programador junior son 2.500 €/mes en España [24], que suele representar el 70 % del coste total [25]. Por lo tanto el coste laboral real serían unos 3.570 €/mes, que son 22,31 €/h.

Teniendo estos factores en cuenta, el coste en horas de trabajo serían
$900 * 22,31 = 20.079$ €

De este modo, los costes de desarrollo totales serían:

Concepto	Coste
Depreciación equipo	76,5 €
Costes laborales	20.079 €
Total	**20.155,50 €**

Tabla 4.2: Estimación de costes de desarrollo

4.5.3. Costes recurrentes

Un precio de hosting más que razonable para la aplicación web sería de 50 €/mes [26], 600 €/año. A esto habría que sumarle unas 2h semanales de tareas de mantenimiento por un técnico, que anualmente serían: $2 * 4 * 12 * 22,31 = 2,141,76\,e$

De este modo, los costes de recurrentes serían:

Concepto	Coste anual
Hosting	600 €
Mantenimiento	2.141,76 €
Total	**2.741,76 €**

Tabla 4.3: Costes anuales recurrentes

Por lo tanto, la estimación de costes totales son 20.155,50 € de desarrollo y 2.741,76 € de gastos recurrentes, visibles en la tabla 4.3.

Capítulo 5

Sistema Desarrollado

A lo largo de este capítulo se explicará el sistema desarrollado en el presente Trabajo de Fin de Grado.

El sistema se compone de un smart contract alojado en la Red B de Alastria y dos aplicaciones que permiten al usuario interactuar con este. La primera se trata de una web que permite a los usuarios interactuar con el smart contract. Además, esta web cuenta con un backend para que los usuarios puedan guardar información para facilitar la interacción con el smart contract. La segunda aplicación para interactuar con el smart contract se trata de un plugin de Moodle, que permitiría a los usuarios de Moodle poder interactuar directamente con el smart contract desde el portal.

La explicación del sistema desarrollado se dividirá en tres partes: análisis del modelo, diseño e implementación. El análisis del modelo se centra en establecer qué funcionalidades se desean implementar, para poder concretar el desarrollo. El diseño se centrará en el diseño de la arquitectura utilizada. Por último, el apartado de implementación detallará cuál ha sido la implementación final de todos los componentes.

5.1. Análisis del modelo

En este apartado nos centraremos en definir qué necesidades queremos cubrir y cuáles son los casos de uso, para pasar a diseñar el sistema.

5.1.1. Historias de usuario

Las historias de usuario son una técnica bastante extendida en el mundo de diseño de software. Ponen su foco en describir los requerimientos de un software desde el punto de vista de un usuario.

Vamos a identificar los 3 tipos de actores principales en el sistema:

- Autenticado: es la persona (o grupo), con un certificado académico.
- Autentificador: es la persona o institución con los credenciales y recursos para poder certificar las credenciales académicas.
- Comprobador: es la persona o institución que quiere comprobar si el certificado académico fue, en efecto, certificado por el autentificador

Teniendo en cuenta los roles definidos, estas serían las historias de usuario teniendo en cuenta los objetivos expuestos en el capítulo 4:

- Como autentificador, querría poder:
  - Emitir certificados digitales de autenticidad de las acreditaciones que otorgo para que se pueda comprobar su veracidad
  - Modificar dichos certificados, para actualizarlos en caso de que halla habido un error o un cambio en su estatus
  - Mantener un historial inmutable de esas certificaciones, para garantizar mayor transparencia y generar confianza
- Como autenticado, mis intereses son:
  - Verificar que el certificado digital no ha sido modificado o alterado desde su expedición para poder utilizarlo en el futuro
  - Poder facilitar una prueba de autenticidad de mis acreditaciones a terceros para que puedan confiar en mis acreditaciones más fácilmente
- Como comprobador, quiero:
  - Verificar si una acreditación cedida por un tercero para saber si es verdadera
  - Verificar si una acreditación ha tenido algún tipo de cambio de estatus por parte de la entidad emisora, para saber si es un cambio relevante

5.1.2. Diagramas de casos de uso

Los diagramas de casos de uso son herramientas fundamentales en el análisis y diseño de sistemas de software, parte esencial del Lenguaje Unificado de Modelado (UML).

Estos ilustran las interacciones funcionales entre los actores del sistema y las aplicaciones desarrolladas, proporcionando una representación visual de cómo los usuarios y las entidades externas interactúan con el sistema. Este diagrama ofrece una visión global de los distintos casos de uso que aborda la aplicación, ayudando a identificar claramente los requisitos de los usuarios y las funcionalidades del sistema que deberán ser implementadas.

La figura 5.2 ilustra los diferentes casos de uso para la web planteada. La figura 5.1 hace lo propio con el plugin de Moodle. Tal y como se puede observar, se ha decidido que los usuarios en los casos de uso puedan ejercer simultáneamente todos los roles definidos en la historia de usuario, ya que genera más transparencia y simplicidad en el sistema.

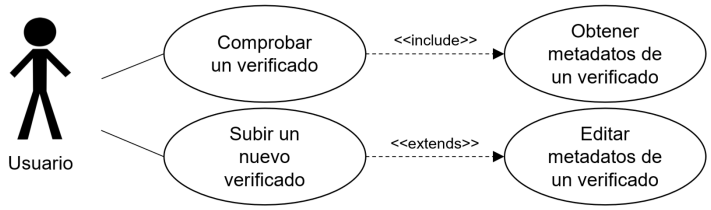


Figura 5.1: Diagrama de casos de uso para plugin de Moodle

5.2. Diseño

Habiendo identificado los requisitos de usuario, podemos pasar a diseñar el sistema propuesto en este Trabajo de Fin de Grado. Para ilustrar correctamente la arquitectura se usarán diagramas de entidad relación para ilustrar el diseño de la base de datos. Para ilustrar el flujo y la interacción entre los componentes del sistema, se usarán diagramas de flujo. Por último, se usará un diagrama de navegación para ilustrar lo que se quiere desarrollar en la interfaz web.

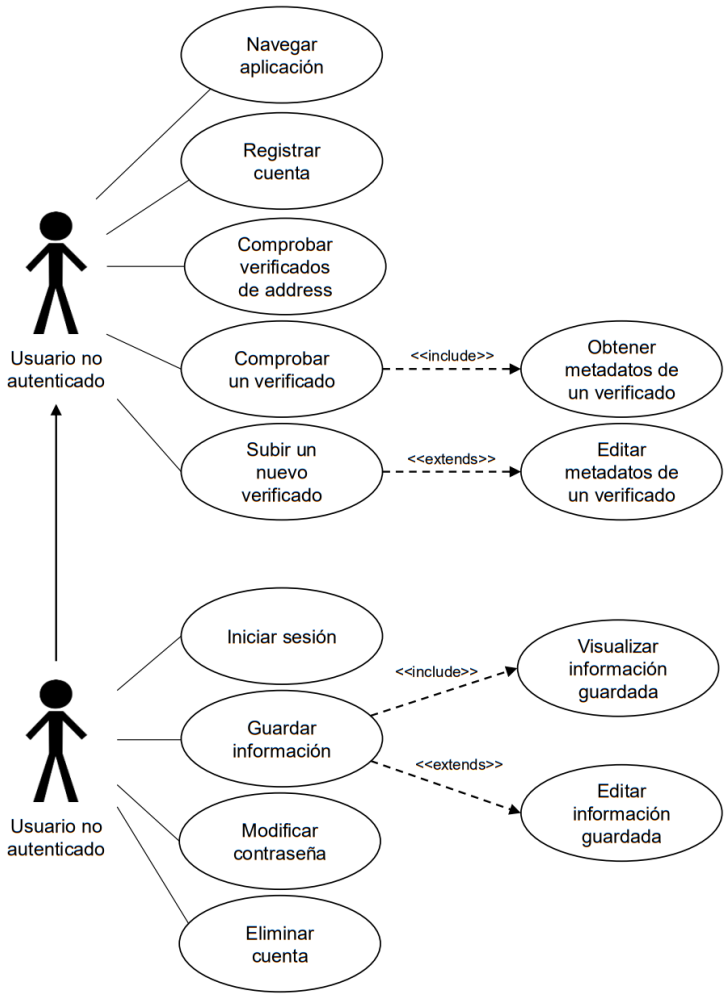


Figura 5.2: Diagrama de casos de uso para aplicación web

5.2.1. Arquitectura

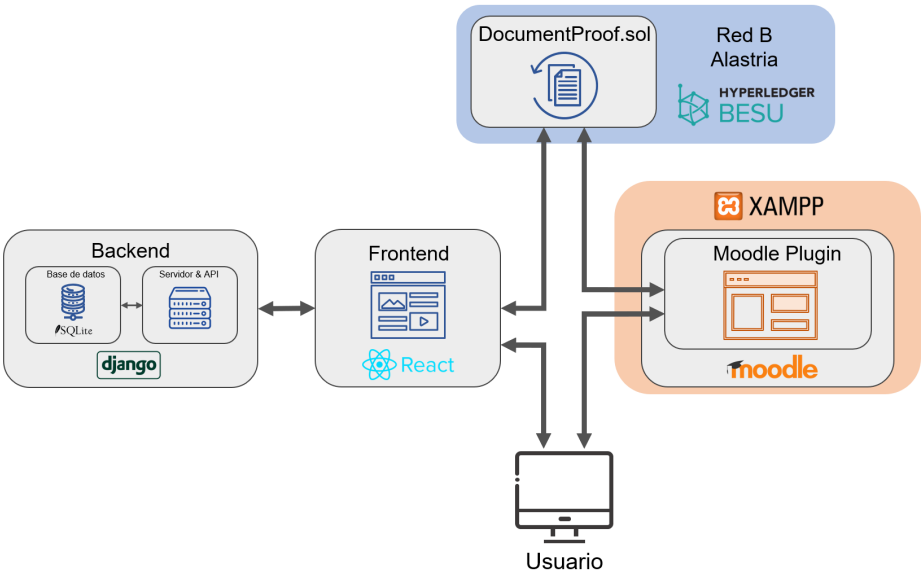


Figura 5.3: Diseño de la arquitectura del sistema

Tal y como se puede observar en la figura 5.3, el diseño de la arquitectura del sistema se compone por tres grandes bloques: la red blockchain (la Red B de Alastria), la web y XAMPP.

Empezando por la red blockchain, se ha elegido la Red B de Alastria, debido a que se trata de una red pública permisionada, contando con las ventajas expuestas en la definición del trabajo. La Red B está construida con Hyperledger Besu, que es un cliente de Ethereum, por lo que el desarrollo de smart contracts es similar al de Ethereum, y también se escriben en Solidity. Cada vez que el sistema desarrollado necesite interactuar con la blockchain, lo hará a través del smart contract, llamado DocumentProof.sol.

El segundo bloque principal es la página web. Se ha decidido separar el backend del frontend lo que aporta modularidad y escalabilidad. Esto facilita la separación de intereses, que es una buena práctica en el desarrollo de software.

Los dos componentes principales de la web serían:

- **Frontend**: se encarga de alojar el frontend de la aplicación web, construido con React 18.1.

  React es una librería JavaScript muy utilizada en el desarrollo frontend por su estructura basada en componentes y sus procesos de renderizado eficientes.

La adopción generalizada de Reacts entre la comunidad de desarrolladores, su completa documentación y su activo ecosistema de bibliotecas de código abierto la hacen una opción idónea para construir un frontend. Entre sus ventajas se encuentra la capacidad de crear interfaces de usuario dinámicas, haciendo que el código sea reutilizable y simplificando las actualizaciones de la interfaz de usuario gracias a su tecnología DOM.

Dado que el frontend se encontraría separado del backend, por las razones ya expuestas, este se comunicará con el backend mediante su API. Del mismo modo, en caso de que el usuario quiera interacturar con blockchain mediante el frontend, este se encargará de comunicarse con el smart contract en la Red B de Alastria. Debido a que el frontend posibilita la interacción con blockchain y el backend al mismo tiempo, es vital que se informe al usuario en todo momento de lo que está haciendo el frontend.

- **Backend**: alojaría el backend, que está basado en Django 4.1.

  Django ha sido elegido gracias a sus características y fiabilidad, que permiten simplificar tareas como la gestión de bases de datos, el manejo de URL y el procesamiento de formularios. Django sigue el patrón Model-View-Controller (MVC), lo que facilita la organización del código y la integración con el frontend. Para gestionar nuestra base de datos de forma eficiente a la vez que aseguramos la integridad y persistencia de los datos, he elegido SQLittle. Se trata de motor de base de datos SQL bien integrado con Django. Esta combinación de Django y SQLittle proporciona una infraestructura escalable que satisface perfectamente los requisitos del proyecto.

  Django se encargaría de gestionar una API que permita al frontend interactuar con este. Esto nos permitirá el usuario pueda crearse cuentas, y guardar la información que hemos determinado en el backend. Además, se va a configurar de tal manera que solo admita llamadas a la API desde el frontend, por motivos de seguridad y eficiencia.

hlEn esta parte podria haber habido solape con definicion de tecnologias

5.2.2. Smart contract

En este apartado nos centraremos en explicar el diseño y funcionalidades del Smart Contract `DocumentProof` en relación con el sistema desarrollado.

Objetivos del Smart Contract

Mientras que la plataforma general tiene la misión de integrar distintos módulos como el frontend, backend y la interacción con Moodle, el Smart Contract específicamente busca proporcionar una prueba inmutable y verificable de la existencia y propiedad de documentos en la blockchain. Esta inmutabilidad garantiza la integridad y verificabilidad del documento a lo largo del tiempo.

Además, las ventajas asociadas a emplear una red como Hyperledger Besu (mayor rapidez y sin coste de gas), permite almacenar más información con facilidad, por lo que hemos decidio incluir una serie de metadatos.

La figura 5.4 ilustra los distintos componentes del smart contract, que se explicarán con más detalle en los siguientes subapartados.

Estructuras de datos

- **Estructura** `Info`: Almacena los metadatos del documento. Sirve como una ficha técnica que complementa el hash único del documento. La información que almacena es:
  - `uint date`: timestamp del bloque donde se guardó el hash por primera vez
  - `uint size`: número pensado para identificar el tamaño del documento guardado, o cualquier otro uso considerado de utilidad por el dueño del hash
  - `string ownerName`: nombre del beneficiario de la acreditación
  - `string title`: título del documento
  - `string description`: descripción del documento
  - `bool isValid`: valor booleano para indicar si el dueño sigue considerando el documento válido
  - `address ownerAddress`: address de la cuenta que subió el hash por primera vez (el propietario)

Mapeos `hashToInfo` y `addressToHashes`: Estos mapeos establecen relaciones clave para el funcionamiento del sistema, vinculando hashes de documentos con sus metadatos y address de propietarios, respectivamente. Además, en el caso de `hashToInfo`, es el mapeo usado para guardar los hashes de documentos.

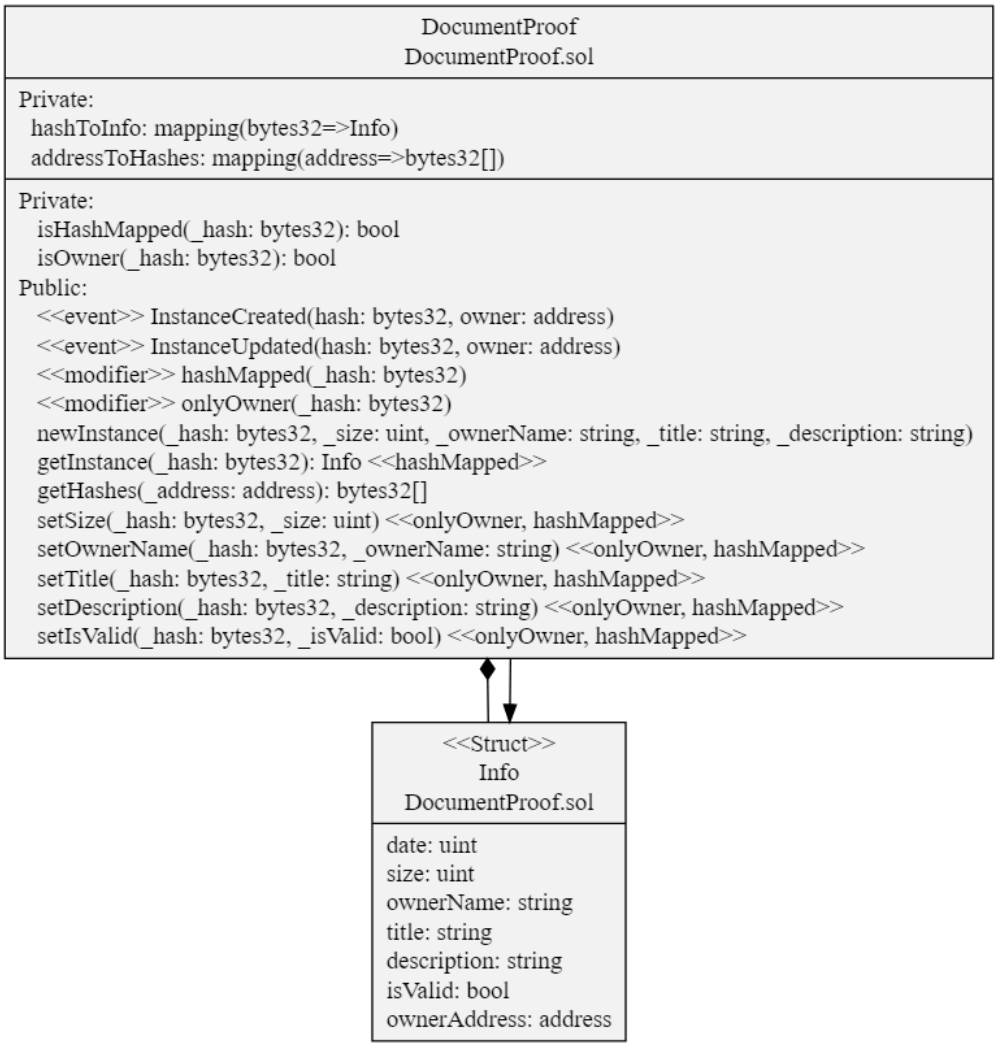


Figura 5.4: Diagrama de clases del smart contract

Diseño de métodos y funcionamiento

Para el diseño de los métodos del contrato hemos tenido dos consideraciones: la primera es que la información almacenada en la estructura Info debería ser accesible para todo el mundo.

La segunda consideración es que hay información que debe ser inmutable, mientras que otra podría ser editada por el propietario del hash. De este modo, la información que consideramos que debería ser inmutable es el propio hash, `date`, `ownerAddress`. De esta forma, garantizamos que el contrato siga funcionando correctamente. El resto de metadatos almacenados en Info podrían ser alterados únicamente por la cuenta que subió originalmente el hash.

De este modo, estos son los principales métodos, modifiers y events del contrato:

- `modifiers`: son similares a funciones, pero sirven para añadirse a funciones y poder modificar su funcionamiento
  - `hashMapped(_hash: bytes32)`: permite saber si un hash ya ha sido guardado en el mapeo `hashToInfo`, que es la manera de registrar los hashes del contrato
  - `onlyOwner(_hash: bytes32)`: permite comprobar si el address que ha llamado al método es el dueño del hash suministrado
- `events`: son mecanismos que permiten registrar logs en la blockchain para ser consultados externamente. Estos serían `InstanceUpdated(hash: bytes32, owner: address)` e `InstanceUpdated(hash: bytes32, owner: address)`, que indicarían cuándo se ha guardado un nuevo hash o cuándo se ha modificado uno existente, respectivamente.
- **Métodos**:
  - **Métodos Privados:** son solo accesibles para otros métodos dentro del contrato.
    - `isHashMapped(_hash: bytes32): bool` - se encarga de la lógica del modifier `hashMapped`
    - `isOwner(_hash: bytes32): bool` - se encarga de la lógica del modifier `onlyOwner`
  - **Métodos Públicos:** pueden ser llamados por cualquiera.
    - `newInstance(_hash: bytes32, _size: uint, _ownerName: string, _title: string, _description: string)` - guarda y mapea un

hash correspondiente a un documento y sus metadatos en una estructura Info.

- `getInstance(_hash: bytes32): Info «hashMapped»` - a partir de un hash, busca su correspondiente Info y la devuelve
- `getHashes(_address: address): bytes32[]` - devuelve los hashes almacenados por una determinada dirección
- `setSize(_hash: bytes32, _size: uint) «onlyOwner, hashMapped»` - permite al owner modificar el size asociado a un hash
- `setOwnerName(_hash: bytes32, _ownerName: string) «onlyOwner, hashMapped»` - permite al owner modificar el ownerName asociado a un hash
- `setTitle(_hash: bytes32, _title: string) «onlyOwner, hashMapped»` - permite al owner modificar el título asociado a un hash
- `setDescription(_hash: bytes32, _description: string) «onlyOwner, hashMapped»` - permite al owner modificar la descripción asociada a un hash
- `setIsValid(_hash: bytes32, _isValid: bool) «onlyOwner, hashMapped»` - permite al owner modificar el apartado isValid asociado a un hash

Conviene matizar además que los métodos `getInstance`, `getHashes`, `isHashMappe` e `isOwner` son de tipo view. Esto significa que no guardan nueva información, sino que se limitan a leerla, lo que permite que no tengan coste en gas, y que por lo tanto el acceso a los mismos sea más rápido, eficiente y barato.

Conclusiones y Relación con Otros Módulos

El Smart Contract `DocumentProof` actúa como el núcleo de la lógica de blockchain de nuestro sistema. Si bien es posible comunicarse directamente con él, ya que la Red B de Alastria es de acceso público, tanto el frontend como el plugin de Moodle implementará esta lógica para acercarlo al usuario final.

En los siguientes apartados, se detallará la implementación de la web, el plugin de Moodle y cómo se integra con este Smart Contract para ofrecer una experiencia fluida y segura a los usuarios.

5.2.3. Plugin de Moodle

Los objetivos que se desean obtener con el plugin de Moodle son:

- Que el usuario pueda generar un hash con un archivo de Moodle
- Poder comprobar si ese hash ha sido subido a la cadena de bloques, y cuáles son sus metadatos
- Poder subir hashes y sus metadatos a la cadena de bloques

En Moodle existen muchos tipos de plugins. El tipo elegido para el Trabajo de Fin de Grado es el de bloques (blocks) debido a que cumple con las caracterísitcas buscadas: es un pequeño bloque funcional que puede actuar como herramienta y mostrar información al usuario. hlcitasr

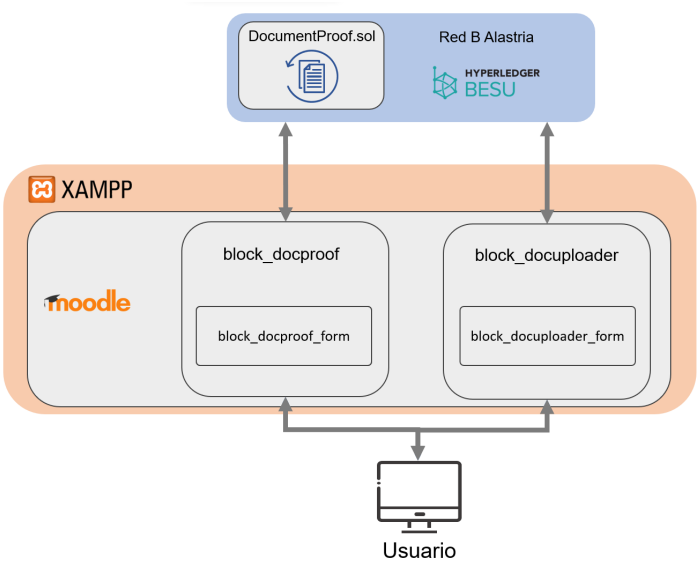


Figura 5.5: Diagrama con principales componentes del Plugin de Moodle

La figura 5.5 muestra cómo se ha diseñado el plugin de Moodle. Estas son las consideraciones más importantes:

- Como queremos que cada bloque sea independiente, cada bloque gestiona por su cuenta la comunicación con el usuario y la red de Alastria
- Para poder acceder al sistema de archivos de Moodle, utilizaremos una clase específica a modo de formulario, encargada de gestionar esa lógica
- La clase principal de cada bloque se ocupará de la lógica de comunicarse con la Red B de Alastria, el usuario y la clase del formulario de archivos

5.2.4. Backend

Tal y como se ha mencionado anteriormente, el diseño del backend se ha basado en Django, que sigue el patrón de diseño Model-View-Template (MVT). El uso de Django ha facilitado en cierta medida la construcción del backend, pues aporta flujos de trabajo y clases para facilitar en gran medida el trabajo de desarrollo.

Explicaremos el diseño comenzando por las estructuras de datos, para pasar a explicar las clases principales y el flujo de datos en el backend.

Estructuras de datos

Para el manejo de la base de datos se ha utilizado SQLite, ya que es notablemente ligero y eficiente. De hecho, en vez de encontrarse en un servidor separado, se encuentra en un archivo al que puede acceder directamente el backend. Si bien para proyectos de mayor envergadura se hacen necesarias soluciones más escalables, para los objetivos de este Trabajo de Fin de Grado es más que suficiente.

Tanto Django como las distintas librerías utilizadas generan sus propias tablas en las bases datos, por lo que en este informe solo haré referencia a aquellas con las que nuestro código interactúa directamente:

- `User`: Es la tabla predefina por Django para la gestión de usuarios. Nosotros solo nos encargamos de gestionar los campos `password`, `username`, `email`, `first_name` y `last_name`; que representan la contraseña, el nombre de usuario, mail, nombre y apellidos, respectivamente.
- `UserHash`: Es la tabla creada para gestionar la información que queremos que guarde el backend. Como simplemente queremos que sirva de ayuda al usuario para guardar hashes, el usuario solo podría introducir hash_value y description. El resto de campos serían el id de cada instancia y user, para poder identificar cada instancia con el user que la ha guardado.

La figura 5.6 muestra los campos de ambas tablas y las relaciones entre sí.

Diseño de clases

La figura 5.7 ilustra la relación entre los 6 componentes principales del backend: urls.py, views.py, models.py, serializers.py, admin.py y la base de datos. Esto es lo que se encarga de hacer cada clase:

Figura 5.6: Modelo de entidad relación de tablas User y UserHash

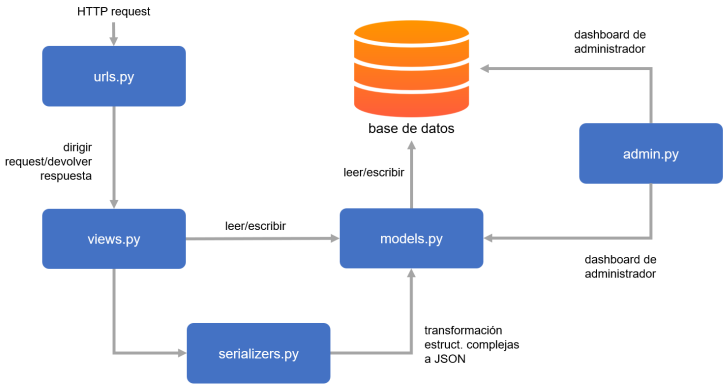


Figura 5.7: Flujo entre los principales componentes del backend

- **Base de datos:** Construida con SQLite e integrada con Django. Se administra a través del sistema de Mapeo Objeto-Relacional (ORM), que permite que sean las propias clases las que hacen consultas.
- **admin.py:** se utiliza para definir la interfaz administrativa para gestionar los datos almacenados en la base de datos de la aplicación a través de un dashboard.
- **models.py:** es donde se define la estructura y el comportamiento de los modelos de datos de la aplicación. Cada modelo se corresponde con una tabla en la base de datos, y el ORM de Django traduce estos modelos en tablas de base de datos.
- **serializers.py:** permiten convertir tipos de datos complejos, como instancias de modelos de Django, en tipos de datos más simples, como JSON. Permiten procesar datos en el contexto de la API.
- **views.py:** El archivo `views.py` contiene la lógica que procesa las solicitudes HTTP entrantes y devuelve las respuestas HTTP correspondientes.
- **urls.py:** El archivo `urls.py` define los patrones de URL para la aplicación, asignando las URL a las views correspondientes.

Autenticación mediante tokens

Tal y como ilustra la figura 5.2 Es necesario que el usuario, desde el frontend, sea capaz de crearse una cuenta, iniciar sesión, ver y editar su información guardada en la base de datos, cambiar su contraseña, y eliminar la cuenta.

La forma elegida para llevarlo a cabo es la autenticación por medio de tokens. La autenticación mediante tokens es un método ampliamente utilizado en el desarrollo de aplicaciones web y API para verificar la identidad de los usuarios. En este contexto, se ha empleado la combinación de las bibliotecas Djoser y SimpleJWT en el marco de Django para implementar esta técnica con eficacia.

Cuando un usuario inicia sesión en la aplicación, el servidor genera un token de acceso único que se proporciona al cliente (por ejemplo, una aplicación móvil o una aplicación web) en lugar de almacenar las credenciales de inicio de sesión en cada solicitud. Djoser, una biblioteca de autenticación y administración de usuarios para Django, simplifica el proceso de registro, inicio de sesión y gestión de cuentas de usuario, lo que resulta en un flujo de autenticación más fluido y seguro.

Por otro lado, SimpleJWT, que es una implementación de JSON Web Tokens (JWT) para Django, permite la generación y validación de tokens de manera se-

gura y eficiente. Los tokens JWT contienen información cifrada sobre el usuario y otros detalles relevantes, como la fecha de expiración del token. Esta información se puede verificar con facilidad, lo que permite a la aplicación garantizar que el usuario tiene acceso válido. La combinación de Djoser y SimpleJWT proporciona una capa de seguridad sólida y una experiencia de usuario mejorada al implementar la autenticación mediante tokens en la aplicación.

El token de autenticación tiene un tiempo de expiración. Pasado dicho tiempo, es necesario que el usuario utilice el toke de refreh para volver a recibir un token de autenticación. Si el token de refresh expira también, le tocará iniciar sesión de nuevo.

La figura 5.8 ilustra cómo funcionaría el proceso de login. El backend se ha abstraído en dos servicios, que representan la lógica de autenticación y la de la interacción con el modelo UserHash. Cuando el usuario inicia sesión, el frontend se encarga de conseguir los tokens de autenticación haciend un API request al endpoint correspondiente. Si consigue recibir los tokens, se encargará de hacer otro request a /api/v1/hashes/ para conseguir la información del usuario y mostrársela.

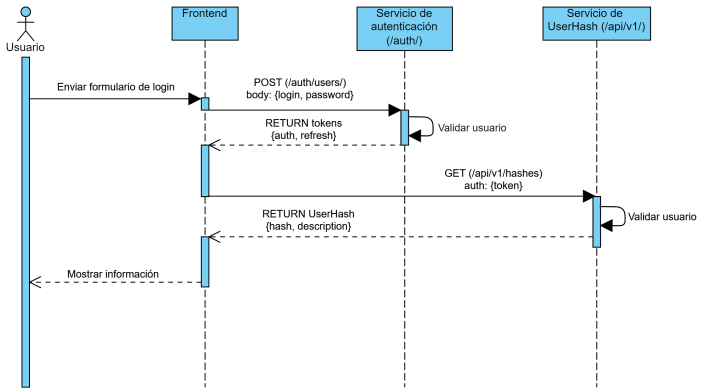


Figura 5.8: Diagrama de secuencia de login: autenticación por tokens

Endpoints de la API

Teniendo en cuenta el uso de la autenticación mediante tokens y los modelelos creados, estos son los endpoints diseñados para la API:

- POST - /auth/token/ → iniciar sesión. Se envían el usuario y la contraseña como parámetros, y en caso de éxito se reciben los tokens de autenticación

y refresh

- POST - /auth/token/refresh/ → generar un nuevo token de autenticación. Hay que enviar el token de refresh como parámetro. En caso de éxito, devuelve un nuevo token de autenticación válido
- POST - /auth/users/ → crear cuenta. Se deben enviar el usuario y contraseña, como mínimo
- DELETE - /auth/users/me/ → eliminar cuenta. Hay que enviar la contraseña como parámetro, además de estar validado.
- POST - /auth/users/set_password/ → cambiar contraseña. Hay que enviar la contraseña actual y la nueva como parámetros, además de estar validado.
- DELETE - /api/v1/hashes/<id>/ → eliminar el id correspondiente a un hash y su descripción en UserHash, si el usuario está autenticado.
- GET - /api/v1/hashes/ → si el usuario está autenticado, devuelve la información que ha guardado en el modelo UserHash.
- POST - /api/v1/hashes/ → permite añadir una nueva entrada a UserHash. El usuario tiene que estar validado e introducir hash_value y la descripción como parámetros.
- PUT - /api/v1/hashes/<id>/ → permite editar la información guardada por un usuario con el determinado <id>. El usuario tiene que estar validad y escribir el nuevo valor de la variable como parámetro.

5.2.5. Frontend

El último elemento del diseño sería el frontend. Para su diseño, tenemos que tener en cuenta los siguientes objetivos:

- Debe ser sencillo e intuitivo, de manera que una persona sin conocimientos de programación pueda usar sin problemas
- Debe permitir a cualquier persona interactuar con el smart contract de una manera sencilla e intuitiva, por lo que debe encargarse de la lógica de la comunicación con la Red B de Alastria
- Debe permitir a cualquier crease una cuenta, iniciar sesión, y poder guardar y editar información en el backend, con el objetivo de que sea más fácil interactuar con hashes

- Como la transparencia y la seguridad son fundamentales para los objetivos de este trabajo de Fin de Grado, es importante que funcione con la máxima transparencia posible, informando al usuario de dónde está exponiendo la información que introduce en la web. Además, es fundamental que aporte información sobre blockchain y los conceptos necesarios para poder entender lo que sucede cuando está interactuando con el smart contract.
- Siempre y cuando no vaya en detrimento de los anteriores objetivos, la aplicación debería ser estética y tener una experiencia de usuario satisfactoria

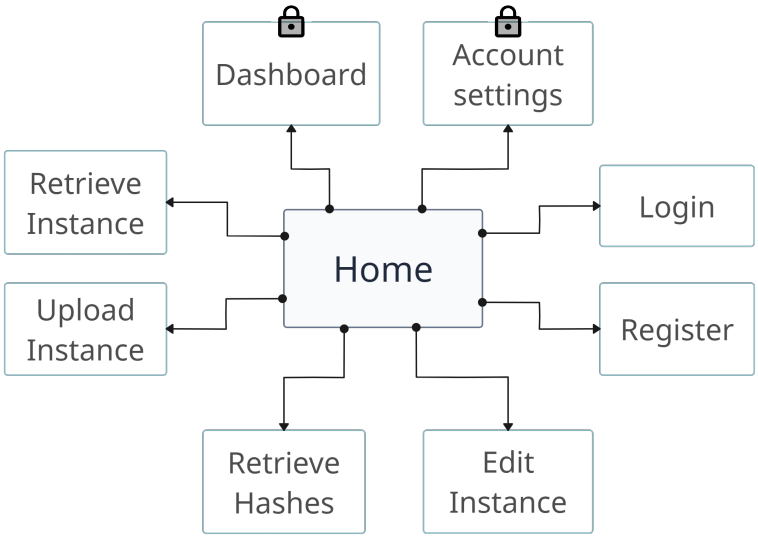


Figura 5.9: Diagrama de vistas del frontend

En la figura 5.9, se puede ver el diagrama con las diferentes vistas propuestas. Aquellas marcadas con un candado solo serían accesibles para usuarios autenticados. Estas son las acciones que podrían llevarse a cabo en cada vista:

- `Home`: bienvenida al usuario e información sobre de qué trata la web, cómo usarla, etc.
- `Retrieve Instance`: dado un archivo, o un hash, notificar al usuario de si este ha sido subido a Blockchain. En caso afirmativo, mostrar la información guardada sobre el mismo.
- `Upload Instance`: permitir que el usuario suba un hash y sus metadatos a blockchain.
- `Retrieve Hashes`: permitir que el usuario obtenga un listado de hashes guardados por una determinada cuenta

- `Edit Instance`: permitir que el usuario edite los metadatos de un hash que ya subió.
- `Login`: iniciar sesión con la cuenta del backend
- `Register`: crear una nueva cuenta en el backend
- `Dashboard`: si el usuario está registrado, mostrar un dashboard con la información guardada en el backend y la posibilidad de editarla o añadir nuevas entradas.
- `Account settings`: si el usuario está registrado, permitirle cambiar su contraseña o eliminar la cuenta.

5.3. Implementación

Después de especificar el diseño de todo el sistema, vamos a pasar detallar las funcionalidades que se han implementado en este Trabajo de Fin de Grado. Se incluirá el código relevante de todos los componentes, excepto el del frontend, por su tamaño. De todas formas, todo el código del presente TFG será accesible en un repositorio de GitHub cuyo link permanente adjuntaré más adelante.

5.3.1. Smart contract

Tal y como se puede ver en el fragmento 5.1, la información se almacena en la estructura `Info` y los dos mappings `hashToInf` y `addressToHashes`.

```
1  pragma solidity 0.8.1;
2
3  contract DocumentProof {
4
5      struct Info{
6          uint date; // timestamp of the block where the hash was
               ↪  stored for the first time
7          uint size;
8          string ownerName;
9          string title;
10         string description;
11         bool isValid; // indicates if the owner still considers
               ↪  the document valid
12         address ownerAddress; // address that uploaded the hash
               ↪  (the owner)
```

```
13      }
14
15      mapping(bytes32 => Info) private hashToInfo;
16      mapping(address => bytes32[]) private addressToHashes;
```

Fragmento de código 5.1: Estructuras de datos de DocumentProof

El siguiene paso en nuestro código sería la definición de eventos y modifiers, visible en el fragmento 5.2. Los modifiers son fragmentos de código que se pueden agregar a otras funciones para modificar su funcionamiento, y así reducir el volumen total de código. En este caso, se encargan de comprobar que otras funciones auxiliares, que explicaré más adelante, devuelven true.

Los eventos van a servir para aprovechar la infraestructura de logging de la red. Su uso es una buena práctica en la programación de Ethereum.

```
1       event InstanceCreated(bytes32 hash, address owner);
2       event InstanceUpdated(bytes32 hash, address owner);
3
4
5       modifier hashMapped(bytes32 _hash){
6           require(isHashMapped(_hash), "Hash is not mapped");
7           _;
8       }
9
10      modifier onlyOwner(bytes32 _hash){
11          require(isOwner(_hash), "Not owner of the hash");
12          _;
13      }
```

Fragmento de código 5.2: Definición de events y modifiers en DocumentProof

El fragmento 5.3 muestra el método `newInstance`, que se encarga de guardar la información en el blockchain, además de comprobar que el hash no haya sido mapeado y de emitir el evento correspondiente

```
1       //records new instance
2       function newInstance(bytes32 _hash, uint _size, string
           ↪ memory _ownerName, string memory _title, string
           ↪ memory _description) public {
```

```
        require(!isHashMapped(_hash), "Hash is already mapped")
            ↪ ;

        hashToInfo[_hash] = Info(block.timestamp, _size,
            ↪ _ownerName, _title, _description, true, msg.
            ↪ sender);
        addressToHashes[msg.sender].push(_hash);

        emit InstanceCreated(_hash, msg.sender);
    }
```

Fragmento de código 5.3: newInstance

Los métodos `getInstance` y `getHashes`, visible en el fragmento 5.4 permiten obtener la información almacenada en un hash y los hashes almacenados por una determinada address, respectivamente.

Es de destacar que, en caso de que el address no haya sido mapeado, devolverá un array vacío. Esta decisión se tomó para garantizar mayor coherencia interna.

Ambos métodos son de tipo view, que significa que no guardan información nueva en la memoria de blockchain. Esto permite que su ejecución sea más eficiente y no tenga un coste de gas.

```
    //returns instance details
    function getInstance(bytes32 _hash) public view hashMapped(
        ↪ _hash) returns (Info memory){
        return (hashToInfo[_hash]);
    }

    //returns hashes stored by an specific address
    function getHashes(address _address) public view returns (
        ↪ bytes32[] memory){
        if (addressToHashes[_address].length != 0){
            return addressToHashes[_address];
        } else {
            bytes32[] memory emptyArray = new bytes32[](0);
            return emptyArray;
        }
    }
```

Fragmento de código 5.4: getInstance y getHashes

En el fragmento 5.5 podemos ver todos los métodos dedicados a permitir al dueño de un hash editar varios de sus metadatos. Todos tienen un funcionamiento similar: incluyen los modificadores `onlyOwner` y `hashMapped`, y se encargan de emitir el evento `InstanceUpdated` en caso de éxito.

```
1     //if called by owner, allows to edit size
2     function setSize(bytes32 _hash, uint _size) public
          ↪ onlyOwner(_hash) hashMapped(_hash){
3         hashToInfo[_hash].size = _size;
4         emit InstanceUpdated(_hash, msg.sender);
5     }
6
7     //if called by owner, allows to edit ownerName
8     function setOwnerName(bytes32 _hash, string memory
          ↪ _ownerName) public onlyOwner(_hash) hashMapped(_hash)
          ↪ {
9         hashToInfo[_hash].ownerName = _ownerName;
10        emit InstanceUpdated(_hash, msg.sender);
11    }
12
13    //if called by owner, allows to edit title
14    function setTitle(bytes32 _hash, string memory _title)
          ↪ public onlyOwner(_hash) hashMapped(_hash){
15        hashToInfo[_hash].title= _title;
16        emit InstanceUpdated(_hash, msg.sender);
17    }
18
19    //if called by owner, allows to edit description
20    function setDescription(bytes32 _hash, string memory
          ↪ _description) public onlyOwner(_hash) hashMapped(
          ↪ _hash){
21        hashToInfo[_hash].description = _description;
22        emit InstanceUpdated(_hash, msg.sender);
23    }
24
25    //if called by owner, allows to edit isValid
26    function setIsValid(bytes32 _hash, bool _isValid) public
          ↪ onlyOwner(_hash) hashMapped(_hash){
27        hashToInfo[_hash].isValid = _isValid;
28        emit InstanceUpdated(_hash, msg.sender);
29    }
```

Fragmento de código 5.5: Métodos para editar metadatos en DocumentProof

Por último, el fragmento 5.6 ilustra los dos únicos métodos privados del smart contract. Estos solo pueden ser llamados por otros métodos del propio smart contract. En nuestro caso, se encargarían de efectuar las comprobaciones mencionadas por los modificadores `onlyOwner` y `hashMapped`. Cabe destacar que en el caso de isHashMapped, esto se hace con el valor de date,.

```
    //returns true if a hash has already been mapped. False
        ↪ otherwise.
    function isHashMapped(bytes32 _hash) private view returns (
        ↪ bool){
        return (hashToInfo[_hash].date > 0);
    }

    //returns true if msg.sender is the owner of an instance,
        ↪ based on the hash. False otherwise.
    function isOwner(bytes32 _hash) private view returns (bool)
        ↪ {
        return (hashToInfo[_hash].ownerAddress == msg.sender);
    }
```

Fragmento de código 5.6: Métodos privados de DocumentProof

Despliegue en red local

Para poder testear el contrato, se desplegó una red local de Hyperledger Besu en una máquina virtual Ubuntu. Esta red constaba de un solo nodo, y sirvió solamente para poder comprobar que el contrato funcionaba correctamente. Estos fueron los pasos seguidos:

1. Creación de la máquina virtual
2. Instalación del software necesario, principalmente: Hyperledger Besu, Truffle, Node.js y npm
3. Configuración del proyecto con Truffle
4. Despliegue de red local de Hyperledger Besu
5. Compilación y despliegue de smart contract con Truffle

Finalmente, varios compañeros de Alastria tuvieron la cortesía de darnos feedback

sobre el smart contract, como último visto bueno antes de desplegarlo definitivamente.

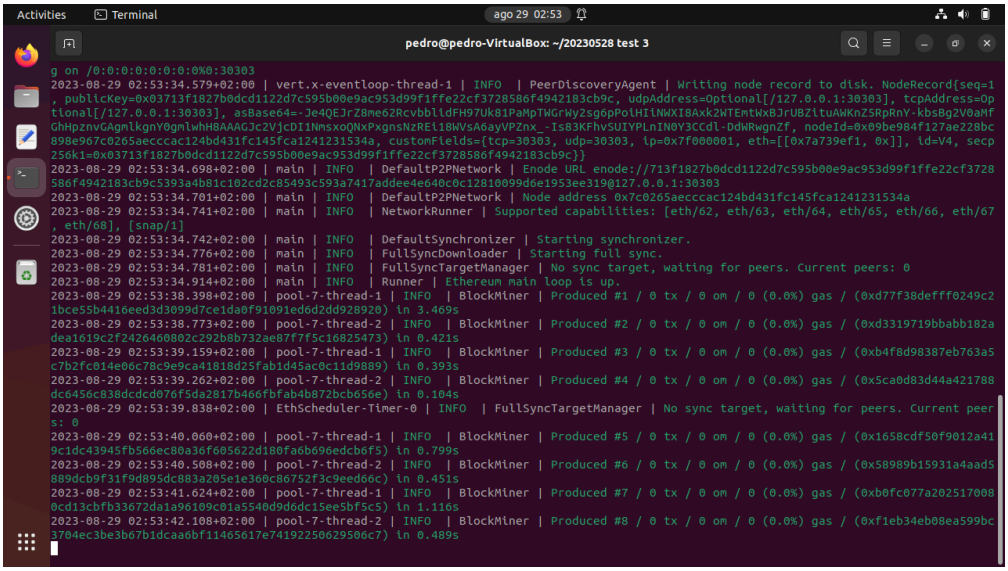


Figura 5.10: Nodo local de Hyperledger Besu ejecutándose en la máquina virtual

Despliegue en Red B de Alastria

A la hora de desplegar definitivamente el contrato, este paso se hizo con Remix IDE. Para ello, hizo falta conectar Metamask a un endpoint público de Alastria, para luego conectarlo con el propio Remix.

La figura 5.11 muestra una captura de Remix habiendo desplegado el contrato, donde se puede ver el formulario para integrar con el mismo de Remix. El fragmento de código 5.7 muestra el recibo de la transacción.

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "result": {
        "blockHash": "0
            ↪ x99cb699658f7e8513e943b99e6c3b1ac9199b37526d211c
        93893627e1a5c3b77",
        "blockNumber": "0x214f8ae",
        "contractAddress": "0
            ↪ xb9a219631aed55ebc3d998f17c3840b7ec39c0cc",
        "cumulativeGasUsed": "0x13fd2e",
        "from": "0xf17f52151ebef6c7334fad080c5704d77216b732",
```

```
        "gasUsed": "0x13fd2e",
        "effectiveGasPrice": "0x0",
        "logs": [],
        "status": "0x1",
        "to": null,
        "transactionHash": "0
            ↪ xfbde34c6d0deef9aa1ad256f5ec84bee2bb58662489fbe13
        0dc00910611f8c77",
        "transactionIndex": "0x0"
    }
}
```

Fragmento de código 5.7: Recibo de la transacción al desplegar DocumentProof en la Red B de Alastria

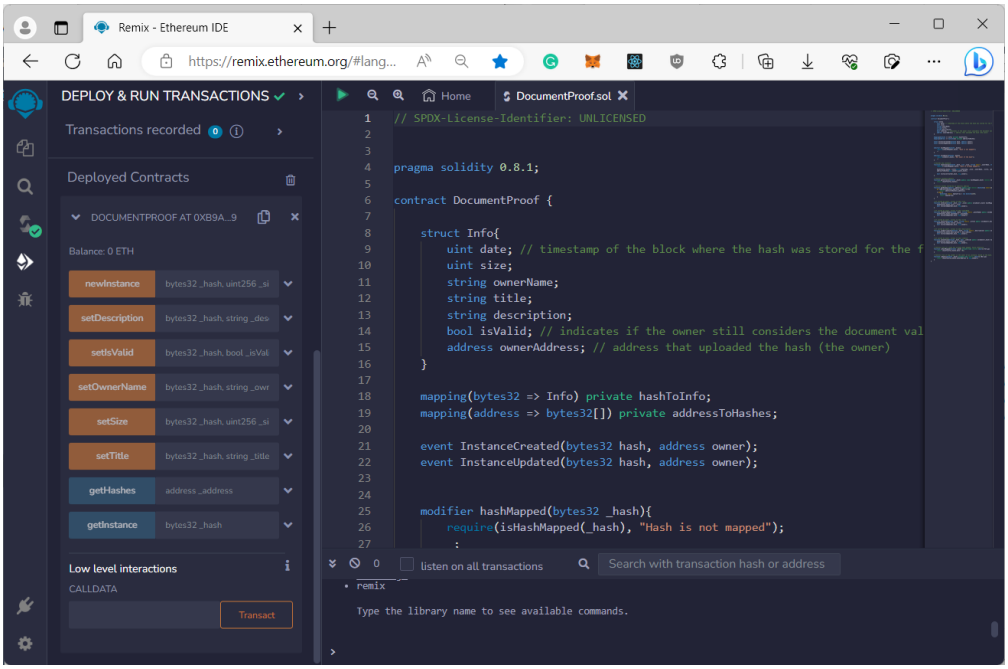


Figura 5.11: Remix IDE con DocumentProof desplegado

5.3.2. Plugin de Moodle

En el caso del Plugin de Moodle, el código de los bloques es relativamente similar, por lo que me limitaré a adjuntar y comentar el código relevante de docproof para poder explicar la implementación de los dos bloques.

```php
class block_docproof_form extends moodleform
{
    protected function definition()
    {
        $mform = $this->_form;
        $mform->addElement('filepicker', 'userfile', '');
        $mform->_elements[0]->_attributes['style'] = "margin: 0
            ↪  auto; display: block;";

        // Form class for uploading the document
        $mform->addElement('submit', 'submitbutton', 'Generate
            ↪ hash');
    }
}
```

Fragmento de código 5.8: Clase de para elegir archivos en Plugin de Moodle

El fragmento 5.8 ilustra cómo se ha creado la clase block_docproof_moodleform, definiendo una nueva clase heredera de `moodleform`. Se trataría de una clase bastante estándar, con la salvedad de que hemos incluido el botón de generar hash.

La clase principal de docproof es una clase hija de `block_base`, que es la clase básica para generar bloques en Moodle. La función más relevante es `get_content()`, pues es la encargada de generar el contenido del plugin. En los siguientes fragmentos de código explicaré los aspectos más importantes de su desarrollo.

```php
        // Initialize the form for uploading the document
        $mform = new block_docproof_form();
        if ($mform->is_submitted()) {
            $file = $mform->get_file_content('userfile');
            if ($file !== false) { // Check if file content
                ↪ exists
                $this->hash = '0x';
                $this->hash .= hash('sha256', $file);
            }
        }
```

Fragmento de código 5.9: Implementación de docproof_form() en docproof

Comenzamos insertando el formulario para elegir archivos. Su clase padre se encarga de gestionar toda la lógica relacionada con elegir el archico y la interfaz de usuario, por lo que nosotros solo nos tenemos que preocupar de generar el hash adecuadamente. Tal y como se puede ver en el fragmento 5.9, en caso de que se haya pulsado el botón de submit con un archivo (comprobación que se hace mediante el último `if`), se genera un hash con el algoritmo SHA256 y se almacena en un variable interna. Por último, conviene añadir que toda esta operación ocurre en el lado del servidor, por lo que luego hará falta implementar la lógica para que el usuario lo pueda ver.

El resto del código del plugin se basa en generar una aplicación en el lado del cliente que se encarga de la lógica de llamar al smart contract mediante la biblioteca Ethers, así como de mostrar otro formulario y la información relevante al usuario. Por motivos de eficiencia, la biblioteca se ha descargado en un archivo aparte, que se debe importar.

Cabe destacar que Ethers necesita de varios datos para poder llamar el contrato:

- Un endpoint para acceder a la red blockchain. En nuestro caso, he usado un endpoint público extraído del GitHub de Alastria.
- El address de nuestro contrato.
- El ABI de nuestro contrato. El ABI es un documento que especifica el funcionamiento del contrato. Se genera cuando este se compila, y en nuestro caso lo extraje con la ayuda de Remix IDE. Se encuentra guardado como un arcchivo independiente en la estructura de archivos del plugin, por lo que es necesario importarlo

En este caso estamos llamando a un método view, que como no necesita de gas para ejecutarse, puede ser accedido directamente por nuestra función. En el caso de docupload, el método al que llamamos no es view, por lo que Ethers se encargará de conectarse a la Red B de Alastria a través de Metamask (debemos tenerlo configurado previamente) para poder firmar la transacción.

Por último, conviene comentar que el script también se encarga de introducir el valor obtenido de hash en el formulario.

```
1 // Constructing the content
2         $this->content->text .= "<p>Please select or upload a
              ↪ file to generate a hash, or type the hash
              ↪ yourself. Press 'Search Hash' to look for the
              ↪ hash in the DocProof smart contract at Alastria's
              ↪  Red B</p>";
```

```
3           $this->content->text .= $mform->render();
4           $this->content->text .= <<<HTML
5               <div class="d-flex justify-content-center align-
                    ↪ items-center" style="flex-direction: column;"
                    ↪ >
6                   <form id="hashForm" onsubmit="event.
                        ↪ preventDefault(); docproof_callContract()
                        ↪ ;">
7                       <input type="text" id="userInputHash"
                            ↪ placeholder="Enter hash here..."
                            ↪ pattern="^0x[a-fA-F0-9]{64}$"
                            ↪ required style="padding: 10px; border
                            ↪ -radius: 4px; border: 1px solid #ccc;
                            ↪  margin-bottom: 10px; margin-top: 10
                            ↪ px;">
8                       <button type="submit" style="padding: 10px
                            ↪ 15px; background-color: #007BFF;
                            ↪ color: white; border: none; border-
                            ↪ radius: 4px; cursor: pointer;">Search
                            ↪  hash</button>
9                   </form>
10              </div>
11          HTML;
12
13          $this->content->text .= <<<HTML
14              <div class="mt-5 d-flex align-items-center justify-
                    ↪ content-center">
15                  <table id="docproof_contractDataTable" style="
                        ↪ display: none; max-width: 45rem;" class="
                        ↪ table table-bordered">
16                      <thead class="thead-dark">
17                          <tr>
18                              <th colspan="2">Stored data
                                    ↪ retrieved from the blockchain
                                    ↪ </th>
19                          </tr>
20                      </thead>
21                      <tbody>
22                          <tr><td>Date</td><td id="
                                ↪ docproof_contractDate"></td></tr>
23                          <tr><td>Size</td><td id="
                                ↪ docproof_contractSize"></td></tr>
24                          <tr><td>Owner Name</td><td id="
                                ↪ docproof_contractOwnerName"></td
                                ↪ ></tr>
25                          <tr><td>Title</td><td id="
                                ↪ docproof_contractTitle"></td></tr
                                ↪ >
26                          <tr><td>Description</td><td id="
```

```
                            ↪ docproof_contractDescription"></
                            ↪ td></tr>
27                      <tr><td>Valid</td><td id="
                            ↪ docproof_contractValid"></td></tr
                            ↪ >
28                      <tr><td>Owner Address</td><td id="
                            ↪ docproof_contractOwnerAddress"></
                            ↪ td></tr>
29                  </tbody>
30              </table>
31          </div>
32          <div id="docproof_errorMessage" class="alert alert-
                ↪ danger mt-3" style="display: none;">Error
                ↪ retrieving data.</div>
33      HTML;
34
35      $abiPath = __DIR__ . '/assets/documentProofABI.json';
36      $contractABI = json_encode(file_get_contents($abiPath))
            ↪ ;
37      $ethersPath = (string) new moodle_url("/blocks/docproof
            ↪ /assets/ethers.min.js");
38
39      $jsCode = <<<EOD
40          <script type="module">
41              import('$ethersPath').then(module => {
42                  window.ethers = module.ethers;
43              });
44
45              document.getElementById("userInputHash").value
                    ↪ = "{$this->hash}";
46
47              window.docproof_callContract = function() {
48                  document.getElementById("
                        ↪ docproof_contractDataTable").style.
                        ↪ display = "none";
49                  document.getElementById("
                        ↪ docproof_errorMessage").style.display
                        ↪  = "none";
50
51                  const userInputHash = document.
                        ↪ getElementById("userInputHash").value
                        ↪ ;
52
53                  var provider = new window.ethers.
                        ↪ JsonRpcProvider('http
                        ↪ ://185.180.8.164:8545');
54                  var contractAddress = "0
                        ↪ xb9a219631aed55ebc3d998f17c3840b7
55                  ec39c0cc";
```

```
56                          var abi = $contractABI;
57
58                          var contract = new window.ethers.Contract(
                                ↪ contractAddress, abi, provider);
59                          contract.getInstance(userInputHash)
60                              .then(result => {
61                                  document.getElementById("
                                        ↪ docproof_contractDataTable").
                                        ↪ style.display = "table";
62                                  document.getElementById("
                                        ↪ docproof_contractDate").
                                        ↪ innerText = new Date(Number(
                                        ↪ result[0]) * 1000).
                                        ↪ toLocaleString();
63                                  document.getElementById("
                                        ↪ docproof_contractSize").
                                        ↪ innerText = Number(result[1])
                                        ↪ .toString();
64                                  document.getElementById("
                                        ↪ docproof_contractOwnerName").
                                        ↪ innerText = result[2];
65                                  document.getElementById("
                                        ↪ docproof_contractTitle").
                                        ↪ innerText = result[3];
66                                  document.getElementById("
                                        ↪ docproof_contractDescription"
                                        ↪ ).innerText = result[4];
67                                  document.getElementById("
                                        ↪ docproof_contractValid").
                                        ↪ innerText = result[5] ? "Yes"
                                        ↪  : "No";
68                                  document.getElementById("
                                        ↪ docproof_contractOwnerAddress
                                        ↪ ").innerText = result[6];
69                              })
70                              .catch(error => {
71                                  let errorMessage = "Error
                                        ↪ retrieving data.";
72
73                                  if (error.message.includes("reason
                                        ↪ =\"Hash is not mapped\"")) {
74                                      errorMessage = "The provided
                                            ↪ hash is not mapped in the
                                            ↪  smart contract.";
75                                  } else if (error.message.includes("
                                        ↪ reverted")) {
76                                      errorMessage = "The smart
                                            ↪ contract method reverted.
                                            ↪ ";
```

```
77                                  } else if (error.message.includes("
                                        ↪ network")) {
78                                      errorMessage = "Network error.
                                            ↪ Please check your
                                            ↪ connection.";
79                                  }
80
81                                  document.getElementById("
                                        ↪ docproof_errorMessage").
                                        ↪ innerText = errorMessage;
82                                  document.getElementById("
                                        ↪ docproof_errorMessage").style
                                        ↪ .display = "block";
83                                  console.error("Error calling the
                                        ↪ contract method:", error);
84                              });
85                  }
86              </script>
87  EOD;
88
89          $this->content->footer .= $jsCode;
```

Fragmento de código 5.10: Construcción del contenido en docproof

5.3.3. Backend

A lo largo de esta sección, me limitaré a explicar los puntos más importantes en el desarrollo del backend para explicar su funcionamiento. No es viable adjuntar todo el código, aunque este se encuentra disponible en el repositorio de GitHub indicado más adelante.

En django, las apps son las estructuras encargadas de implementar una funcionalidad. En nuestros caso, la app creada para encargarse de nuestros modelos y la API se llama “api”.

Configuración en settings.py

En primer lugar, es necesario añadir los plugins relevantes al archivo de configuración "settings.py", tal y como se puede ver en el fragemento 5.11, en la parte correspondiente a “INSTALLED_APPS”.

```
1
2  # Importing relevant apps
3  INSTALLED_APPS = [
4      ...
5      'rest_framework',
6      'api', # Nuestra aplicacion principal
7      'djoser', # Plugin para generar un backend con
           ↪ funcionalidades de autenticacion
8      'corsheaders', # Plugin para restringir que IPs pueden
           ↪ llamar a nuestra API
9      'rest_framework_simplejwt', # Plugin para generar tokens de
           ↪  autenticacion
10     'rest_framework.authtoken', # Permite gestion de tokens de
           ↪ autenticacion
11 ]
12
13 REST_FRAMEWORK = {
14     'DEFAULT_AUTHENTICATION_CLASSES': (
15         'rest_framework_simplejwt.authentication.
               ↪ JWTAuthentication', # EStablecemos clases de
               ↪ autenticacion
16     ),
17 }
18
19 # Configuracion de simple_jwt
20 SIMPLE_JWT = {
21     'ROTATE_REFRESH_TOKENS': False,
22     'ALGORITHM': 'HS256',
23     'SIGNING_KEY': SECRET_KEY,
24     'VERIFYING_KEY': None,
25     'AUTH_HEADER_TYPES': ('Bearer',),
26     'USER_ID_FIELD': 'id',
27     'USER_ID_CLAIM': 'user_id',
28     'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.
           ↪ AccessToken',),
29     'TOKEN_TYPE_CLAIM': 'token_type',
30 }
31
32 # Nuestra base de datos
33 DATABASES = {
34     'default': {
35         'ENGINE': 'django.db.backends.sqlite3',
36         'NAME': BASE_DIR / 'db.sqlite3',
37     }
38 }
39
```

```
# Configurando CORS
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000", # React frontend
]
```

Fragmento de código 5.11: Fragmentos de la configuracion en settings.py

Este es un resumen de los distintos plugins usados y su propósito:

- `rest_framework`: Proporciona herramientas para construir API web con Django.
- `api`: Nuestra aplicación principal.
- `djoser`: Plugin para generar un backend con funcionalidades de autenticación.
- `corsheaders`: Plugin para restringir qué IPs pueden llamar a nuestra API.
- `rest_framework_simplejwt`: Plugin para generar tokens de autenticación.
- `rest_framework.authtoken`: Permite la gestión de tokens de autenticación.

Además, en settings.py hemos definido la configuración para varias de estos plugins y la base de datos, tal y como está indicado en el fragment 5.11.

models.py

El fragmento 5.12 muestra la definición de nuestro model UserHash, que es el encargado de guardar la información del usuario. La variable `unique_together` especifica que cada user no puede guardar más de una vez el mismo hash. Además, `description es opcional`.

No hace falta declarar el modelo User porque es un modelo creado por defecto por Django.

```
from django.db import models
from django.contrib.auth.models import User

class UserHash(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    hash_value = models.CharField(max_length=256)
```

```
    description = models.TextField(blank=True, null=True) #
        ↪ Optional description of the hash

    class Meta:
        unique_together = ('user', 'hash_value')
```

Fragmento de código 5.12: models.py

serializers.py

La clase serializers.py se encarg de configurar los serializers, tal y como se explicó en el diseño del backend. En nuestro caso, solo hace falta crear el serializer de `UserHash`, donde especificamos que üser"no es un apartado obligatorio a la hora de crear nuevas instancias llamando a la API, ya que queremos que de eso se encargue el servidor internamente en views.py.

Con respecto a User, creamos un Serializer a partir del creado por defecto para instanciar una clase Meta, que se encargaría de definir cuáles son los campos por defecto de User.

```
from rest_framework import serializers
from .models import UserHash
from djoser.serializers import UserCreateSerializer


class CustomUserCreateSerializer(UserCreateSerializer):
    class Meta(UserCreateSerializer.Meta):
        fields = ("id", "email", "username", "first_name", "
            ↪ last_name", "password")


class UserHashSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserHash
        fields = "__all__"
        read_only_fields = (
            "user",
        )  # Exclude 'user' from required fields during
            ↪ creation
```

Fragmento de código 5.13: serializers.py

views.py

Tal y como se muestra en el fragment 5.14, solo es necesario especificar las vistas relacionadas con extraer la información de UserHash, ya que djoser ya se ecnarga de implementar las relacionadas con la autenticación.

Los puntos más relevantes de la configuración de views.py son:

- Especificamos cuál es el serializer, la clase encargada de las autenticaciones, los permisos de acceso y a qué queryset da acceso
- Especificamos que solamente se puede extraer información del usuario que está autenticado
- Como en serializers.py especificamos que no había que proporcionar el User al usar la API, aquí implementamos la lógica para configurar que `UserHasViewSet` se encargue de obternerlo.

```
1  from rest_framework import viewsets
2  from .models import UserHash
3  from .serializers import UserHashSerializer
4  from rest_framework.permissions import IsAuthenticated
5  from rest_framework.permissions import IsAuthenticated
6  from rest_framework_simplejwt.authentication import
       ↪ JWTAuthentication
7
8
9  class UserHashViewSet(viewsets.ModelViewSet):
10     serializer_class = UserHashSerializer  # Use the serializer
           ↪  we created
11     authentication_classes = [JWTAuthentication]  # Use JWT for
           ↪  authentication
12     permission_classes = [IsAuthenticated]  # Only
           ↪ authenticated users can access
13     queryset = UserHash.objects.all()  # Get all the hashes
14
15     def get_queryset(self):
16         return UserHash.objects.filter(
17             user=self.request.user
18         )  # Get only the hashes for the current user
19
```

```
20      def perform_create(self, serializer):
21          serializer.save(
22              user=self.request.user
23          )  # Save the user when creating a new hash
```

Fragmento de código 5.14: views.py

urls.py

La figura 5.15 muestra los urls.py de todo el backend, y de la aplicación `api` (la aplicación principal)

De esta configuración cabe destacar:

- la creación de todos los endpoints relacionados con el manejo de la información en nuestra api se han agrupado en `/api/v1/`. A partir de allí, hemos utilizado la extensión rest_framework para generar automáticamente las URLS de nuestra api.
- hemos asignado el endpoint para cargar el dashboard de admin a `/admin/`
- hemos usado djoser y jwt para configurar los endpoints relacionados con la autenticación y el manejo de tokens.

```
1  #urls.py of api app
2  from django.urls import path, include
3  from rest_framework.routers import DefaultRouter
4  from .views import UserHashViewSet
5
6  router = DefaultRouter() # Create a router. This will create
       ↪ all the urls for us automatically
7  router.register(r'hashes', UserHashViewSet) # Register the
       ↪ viewset with the router
8
9  urlpatterns = [
10     path('', include(router.urls)), # Include the router urls
11 ]
12
13 ########################################
14 # ursl.py of Django project
15
16 from django.contrib import admin
17 from django.urls import path, include
```

```
18 from rest_framework_simplejwt.views import TokenObtainPairView,
       ↪  TokenRefreshView
19
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('api/v1/', include('api.urls')),
24     path('auth/', include('djoser.urls')),
25     path('auth/', include('djoser.urls.jwt')),
26     path('auth/token/', TokenObtainPairView.as_view(), name='
           ↪ token_obtain_pair'),
27     path('auth/token/refresh/', TokenRefreshView.as_view(),
           ↪ name='token_refresh'),
28 ]
```

Fragmento de código 5.15: Los 2 urls.py

5.3.4. Frontend

El frontend se ha creado utilizando Node.js y npm. El primero es un entorno de ejecución que permite ejecutar JavaScript en el ordenador, mientras que npm es un getor de paquetes, que permite gestionar el sistema de archivos de manera cómoda y eficiente.

Como he comentado anteriormente, el frontend ha sido construido con React. Además de React, hemos utilizado el framework Material Design for Bootstrap (MDB), que es un framework con numerosos componentes y estilos ya incluidos, para simplificar la generación de webs estéticas y con una buena experiencia de usuario.

Como la cantidad de código es demasiado como para adjuntarla, estrucutraré esta sección a través de 3 secciones: components, interacción con API, e interacción con smart contract.

Descripción de components

Los components son uno de los conceptos fundamentales de React. Se tratan de fragmentos de la interfaz de usuario autocontenidos. Cada uno puede encargarse independientemente de su código de javasript y de su HTML. Esto significa que, en términos generales, son elementos independientes que se pueden adjuntar a lo largo

de la web. Además, un componente puede anidar componentes hijos e interactuar con ellos.

MDB trae sus propios componentes. Además, se ha desarrollado una serie de componentes y funciones para implementar el frontend:

- Components responsables de vistas: son los componentes primarios de cada vista del frontend
  - App: encargado de configurar toda la página, la barra de navegación y los headers
  - Home: vista de home
  - Login: vista de login
  - Register: vista de register
  - Dashboard: vista del dashboard de información guardada para usuarios registrados
  - Account: vista de ajustes de cuenta para usuarios regsitrados
  - GetInstance: vista para comprobar un hash en blockchain
  - GetHashes: vista para recuperar hashes subidos por un address a la blockchain
  - EditInstance: vista para editar los metadatos de un hash en blockchain
  - Upload: vista para subir nueva información a blockchain
- Componentes auxiliares: son componentes llamados de manera auxiliar dentro de las vistas
  - DataTable: vista auxiliar para generar la tabla en Dashboard
  - NotLoggedInComponent: genera un output auxiliar en caso de que el usuario no esté registrado y necesite estarlo
  - LoadingComponent: genera un output auxiliar de "cargando"
  - AuthContext: genera un contexto de autenticación. Lo explicaré con más detalle más adelante
  - HashGenerator: formulario para generar un hash a partir de un archivo
- Funciones: archivos de funciones de javascript para integrar funcionalidades

- useHashes: hook (función que permite modificar el estado de componentes), para obtener los datos de Dashboard
- useProtectedRoute: hook para redirigir al usuario a Dashboard si ya está autenticado
- apiService: funciones para interactuar con la API del backend

La figura 5.12 muestra las dependencias entre componentes, a la hora de entender cómo se distribuyen en la web.

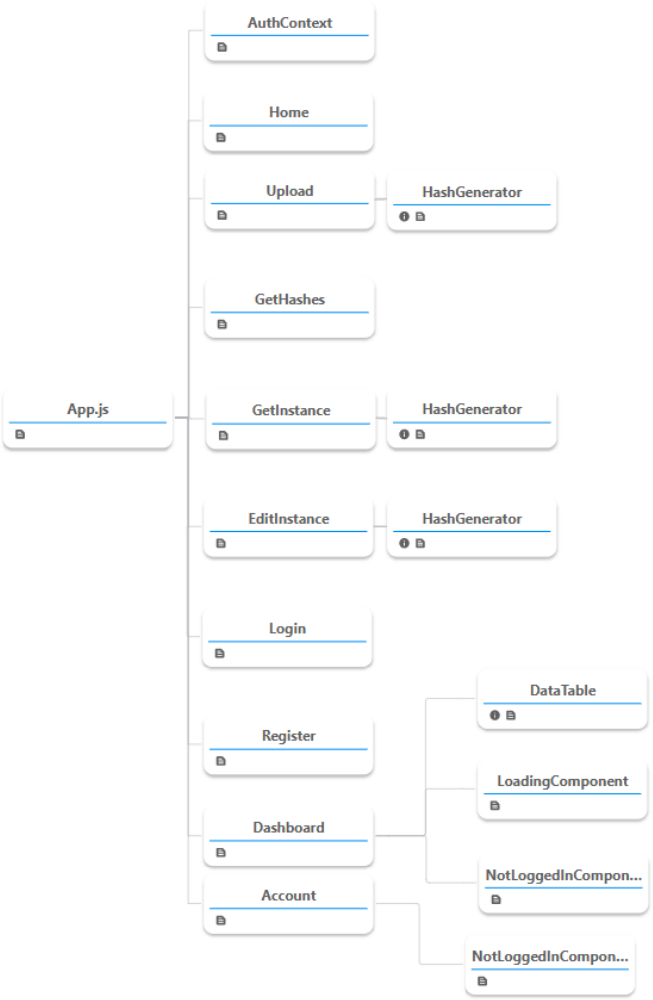


Figura 5.12: Árbol de componentes de React del frontend

Autenticación

Para gestionar la autenticación se ha usado el componente AuthContext visible en el fragmento 5.16. Un contexto en React es una manera de enviar propiedades a componentes anidados y a todos sus hijos, sin que haya necesidad de irlos pasando manualmente. Esto me ha permitido garantizar que todas los componentes tuvieran acceso a la información adecuada. AuthContext encapsula todas las vistas, por lo que todas pueden acceder a los valores que pasa a sus hijos.

Los elementos clave para comprenderlo son:

- La función setAuthDetails guarda en la memoria local los tokens. Luego pueden ser accesibles por la App.
- Las funciones handlelogout y logout permiten cerrar sesión, eliminando la información correspondiente
- El hook [currentUser, setCurrentUser] permite saber qué usuario está autenticado

```
import React, { createContext, useState, useContext, useEffect } from "react";

const AuthContext = createContext();

export const useAuth = () => {
  return useContext(AuthContext);
};

export const AuthProvider = ({ children }) => {
  useEffect(() => {
    // This useEffect will run only once, when the component is mounted
    const token = localStorage.getItem("access_token");
    const savedUsername = localStorage.getItem("username");
    if (token && savedUsername) {
      setCurrentUser(savedUsername);
    }
  }, []);

  const [currentUser, setCurrentUser] = useState(null);

  const setAuthDetails = (user, tokens) => {
    setCurrentUser(user);
    localStorage.setItem("username", user); // save username
```

```
24      localStorage.setItem("access_token", tokens.access); //
            ↪ save access token
25      localStorage.setItem("refresh_token", tokens.refresh); //
            ↪ save refresh token
26    };
27
28    const logout = () => {
29      setCurrentUser(null); // This clears the current user
30      localStorage.removeItem("access_token");
31      localStorage.removeItem("refresh_token");
32    };
33
34    const handleLogout = (e) => {
35      e.preventDefault(); // prevent default behavior
36      e.stopPropagation(); // stop event propagation
37
38      logout();
39    };
40
41    return (
42      <AuthContext.Provider
43        value={{
44          currentUser,
45          setAuthDetails,
46          setCurrentUser,
47          logout,
48          handleLogout,
49        }}
50      >
51        {children}
52      </AuthContext.Provider>
53    );
54  };
```

Fragmento de código 5.16: Ejemplo de uso de axios para interactuar con API

Interacción con API

Para la interacción con la API del backend se ha utilizado la librería `axios`. El fragmento de código 5.16 muestra cómo se ha usado para interactuar con la API.

```
1
2  const fetchHashes = async () => {
```

```
  try {
    const response = await axios.get("/api/v1/hashes/", {
      headers: authHeader(),
    });
    return response.data;
  } catch (error) {
    throw error;
  }
};
```

Fragmento de código 5.17: Ejemplo de uso de axios para interactuar con API

Interacción con Smart Contract

Para interactuar con el smart contract, se ha utilizado la librería `ether` de manera análoga al plugin de Moodle, aunque en este caso se puede interactuar con todas los métodos del contrato.

```

const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const provider = new ethers.JsonRpcProvider(PROVIDER_URL)
        ↪ ;
      const contract = new ethers.Contract(
        CONTRACT_ADDRESS,
        documentProofABI,
        provider
      );

      // Since getInstance is a view/call method:
      const result = await contract.getHashes(formData);
      console.log("Result from contract:", result);
      setContractData(result);

      // No need to wait since it's not a transaction
    } catch (error) {
      console.error("Error fetching contract data:", error);
      alert("Error fetching contract data: " + error.message);
    }
  };
```

Fragmento de código 5.18: Ejemplo de uso de ether para interactuar con smart contract

Capítulo 6

Análisis de resultados

Después de desarrollar e implementar el sistema especificado en el capítulo anterior, se han conseguido alcanzar todos los objetivos principales.

Este capítulo se dividirá en 3 apartados, correspondientes al backend, frontend, y plugin de Moodle.

No es posible incluir todo el código del presente trabajo de fin de grado en esta memoria, pero lo he subido a GitHub (link: https://github.com/Pedro-Cuevas/tfg-icai).

6.1. Backend

Tal y como se ha explicado en el capitulo anterior, el backend integra, a través de la RESTful API, los endpoints necesarios para que se pueda hacer uso del mismo.

Se ha comprobado, tal y como ilustra la figura 6.1, el funcionamiento de todos los endpoints por medio de Postman.

Asimismo, el propio Django permite generar y personalizar una página de administrador, que permite interactuar directamente con las bases de datos, visible en la figura 6.2.

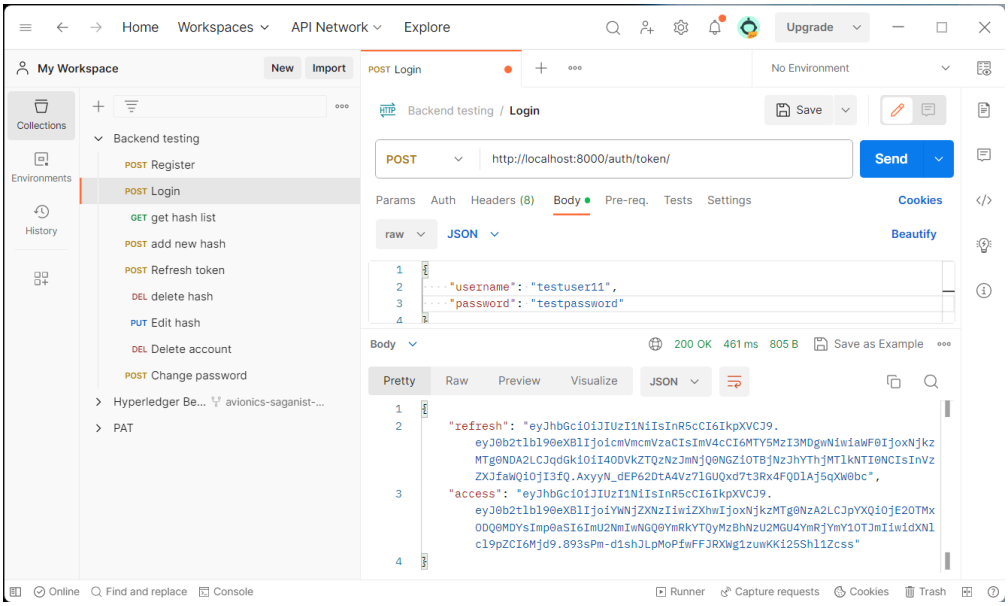


Figura 6.1: Uso de Postman para usar la API del backend

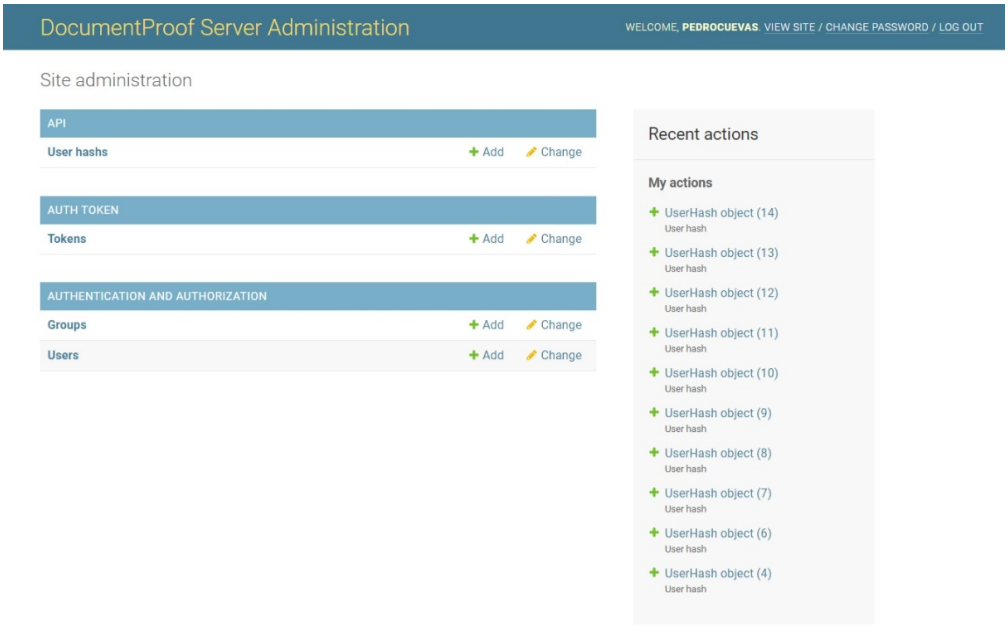


Figura 6.2: Página de administración del backend personalizada

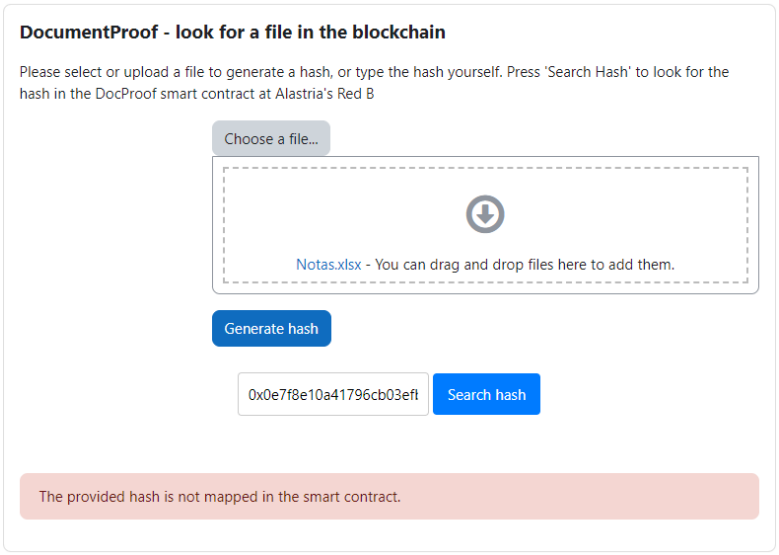


Figura 6.3: Plugin de Moodle: ejemplo al introducir un hash no reconocido

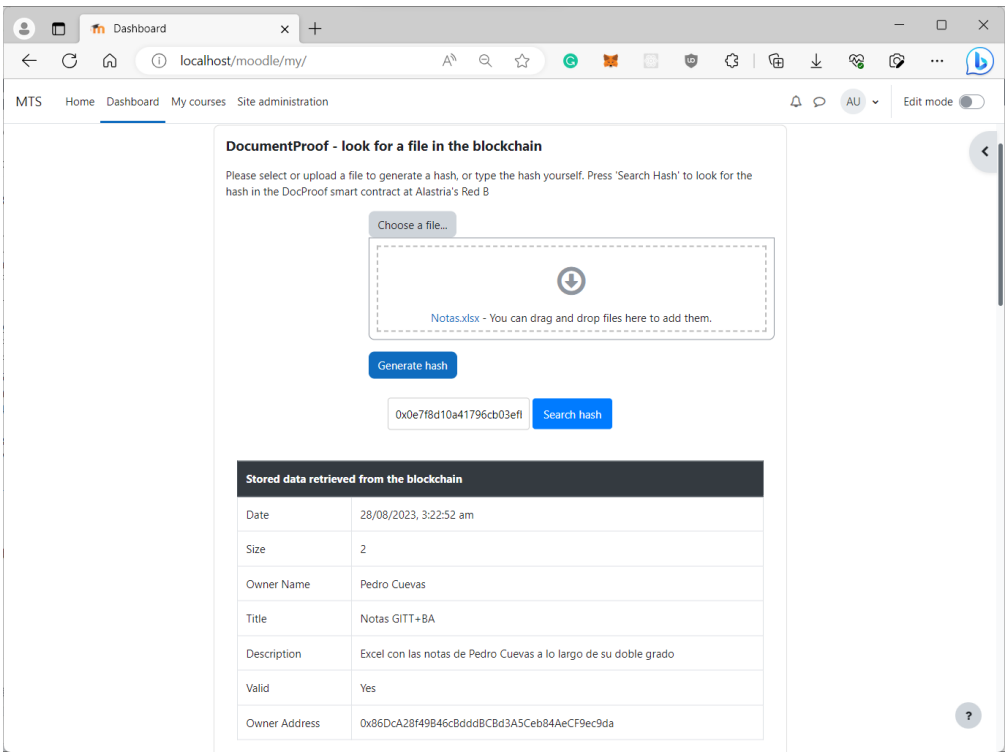


Figura 6.4: Plugin de Moodle: ejemplo al recuperar información de blockchain

6.2. Plugin de Moodle

El plugin de Moodle se compone de dos blocks totalmente independientes: docupload y docproof. El primero sirve para autenticar archivos y el segundo para comprobarlos. Además, ambos ofrecen la posibilidad de elegir un archivo del propio sistema de archivos de Moodle, para generar un hash a partir del mismo.

Comenzando por docproof, este tienen una interfaz más simple: el usuario puede introducir manualmente un hash, o utilizar el formulario de Moodle para elegir un archivo y generar el hash a partir del mismo. A partir de ahí, al pulsar el botón de "Search hash", el plugin intentará ponerse en contracto con la red blockchain, y buscará el hash. Ofreciendo al usuario la información del mismo si lo encuentra (figura 6.4) o enviando un mensaje de error 6.3).

Conviene añadir además, que el formulario se encarga de verificar que el contenido tecleado en el formulario tenga formato de hash.

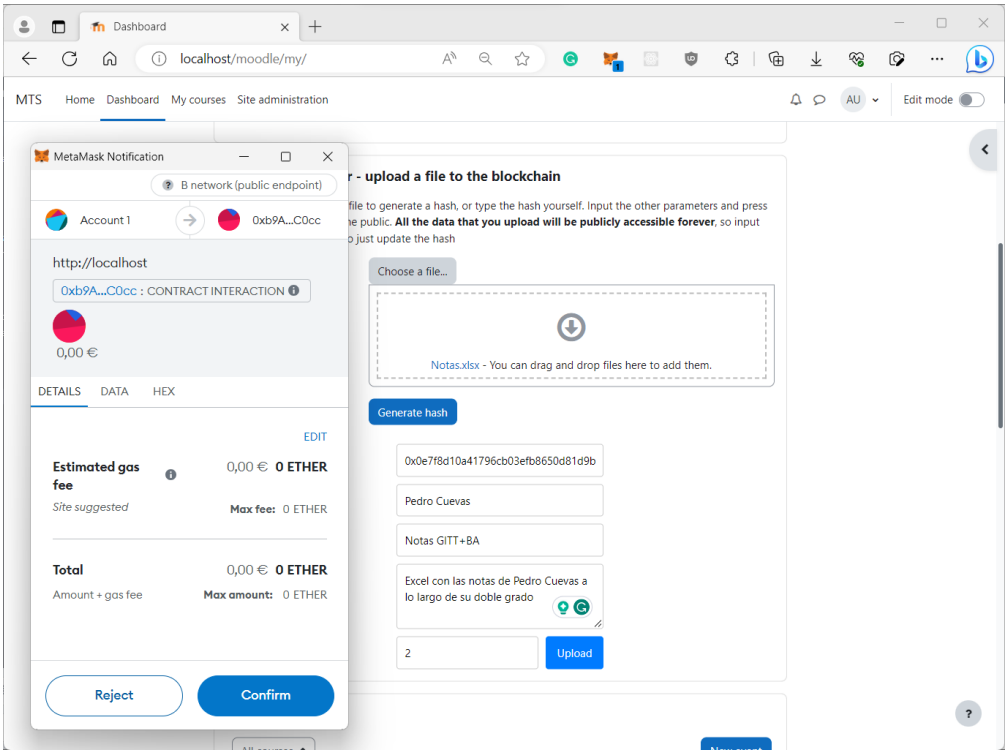


Figura 6.5: Pluigin de Moodle: ejemplo al subir información a blockchain

En el caso de docupload, este es un poco más complejo, debido a la naturaleza de la operación que se quiere realizar. Tal y como se puede ver en la figura 6.5, este

cuenta con un formulario más largo, que además también comprueba el formato de los datos introducidos. Si estos son válidos, intentará procesar la transacción (informando al usuario si ocurre algún problema), para lo que abrirá Metamask (encargado de firmar la operación), tal y como se puede ver en la figura.

6.3. Frontend

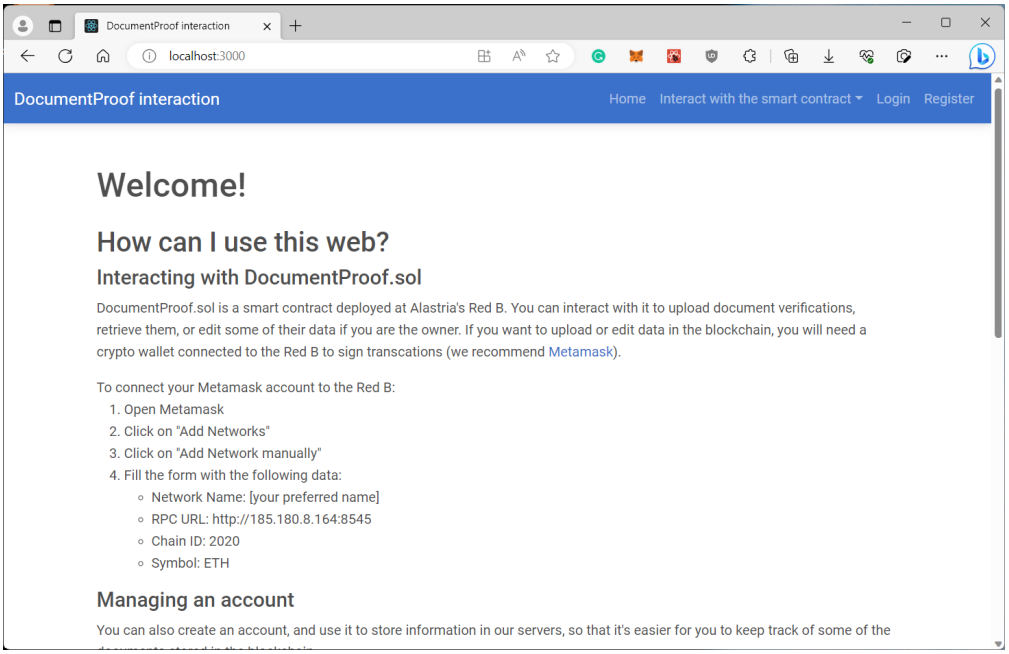


Figura 6.6: Pantalla home

Los nombres de las vistas son los especificados en la figura 5.9. La primera vista al acceder a la web es la mostrada en la figura 6.6. En ella, se da la bienbenida al usuario, además de información básica sobre la web, cómo conectarse con Metamask a la red B de Alastria, etc.

La navegación dentro de la web se hace principalmente mediante la barra de navegación en la parte superior, que está presente en todas las vistas. El botón "Interact with the smart contract" es un desplegable que lleva a las 4 vistas relacionadas con la interacción con el smart contract, que explicaré a continuación.

Las vistas representadas en las figuras 6.7 y 6.8 muestran la vista "Retrieve Instance". Esta se compone for 3 elementos funcionales: una tarjeta, que ofrece información al usuario, un formulario para introducir un hash, o generarlo a partir de

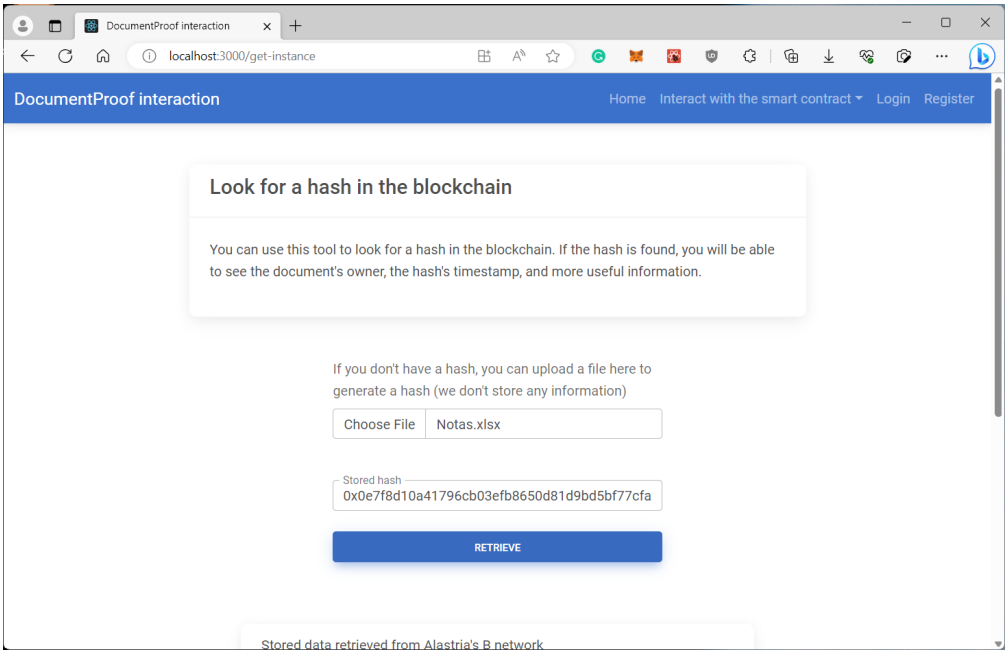


Figura 6.7: Parte superior de vista getInstance

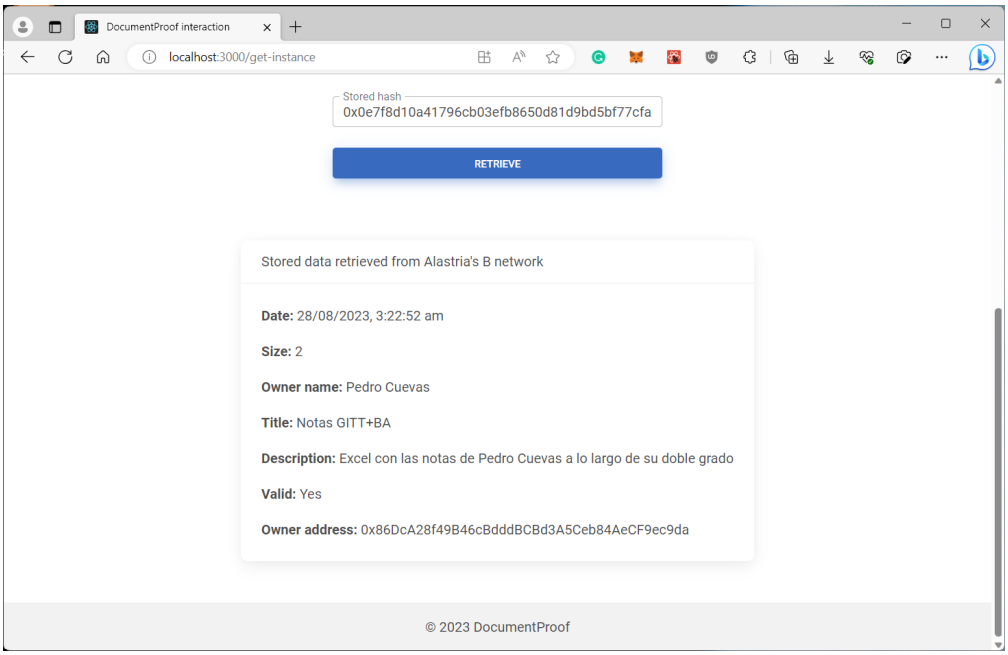


Figura 6.8: Parte inferior de vista Retrieve Instance

un archivo, y el output, que aparecerá en caso de encontrarse el hash en blockhain. En caso de error, se alerta al usuario con información específica dependiendo del tipo de error.

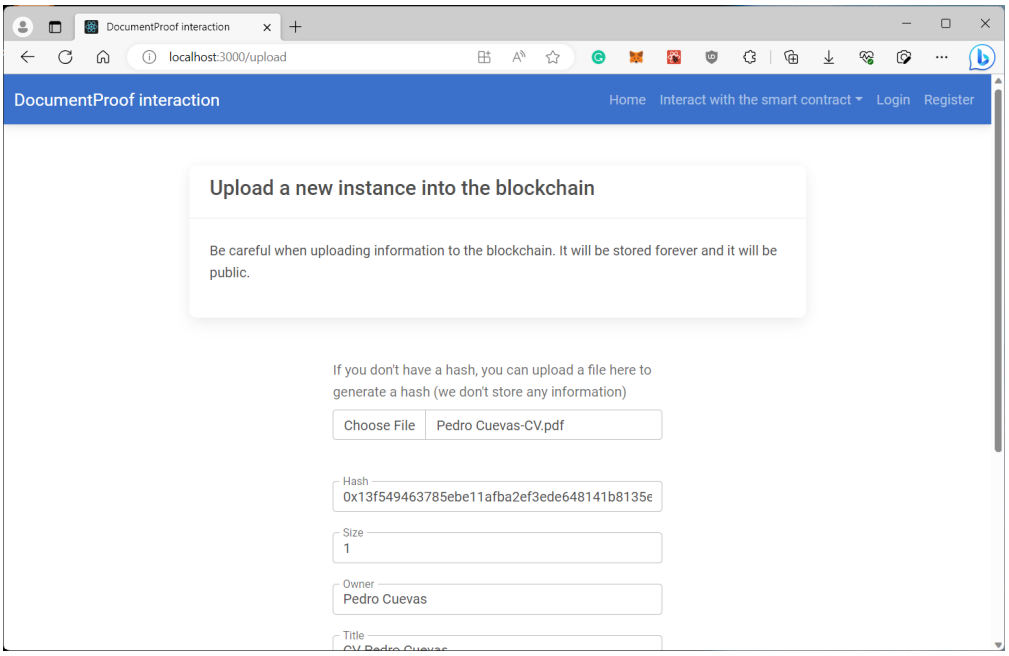


Figura 6.9: Parte superior de vista Upload

Las figuras 6.9 y 6.10 muestran la vista "Upload", que funciona de una manera totalmenta análoga a la ya descrita en el plugin de Moodle. Si se produce un error, o el usuario consigue subir de manera exitosa el hash y los metadatos, se le notificará a través de una alerta.

Asimismo, cabe detacar que el formulario también implementa controles adecuados para asegurar que el usuario rellena todos los campos con el formato adecuado. La tarjeta de información alerta al usuario de que todo lo que suba a blockchain será accesible para todo el mundo, para sensibilizar al usuario al respecto.

La figura 6.11 muestra la vista "Retrieve Hashes". Cuando el usuario introduce un address y pulsa el botón, el frontend se encarga de recoger la información correspondiente y adjuntarla en la tabla inferior, notificando al usuario si la cuenta no tiene hashes registrados o se produce un error de cualquier tipo. Cada hash es además un link que lelva a la vista "Get Instance", con el propio hash ya introducido en el formulario.

Las figuras 6.12 y 6.13. Muestran la vista "Edit Instance". Esta vista se compone de dos partes principales. La primera funciona de una manera similar a "Get Ins-

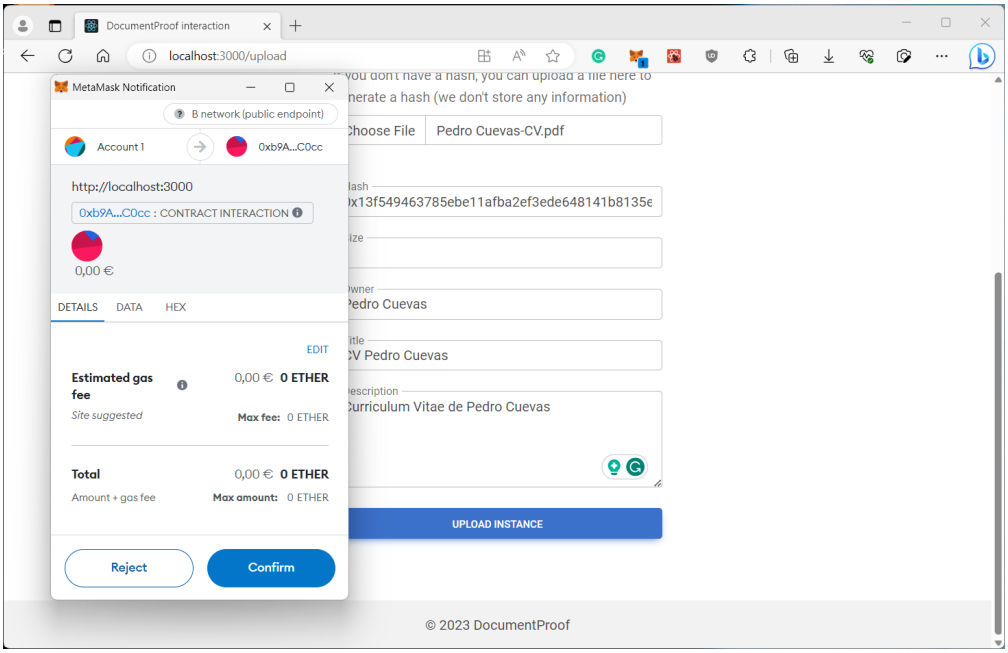


Figura 6.10: Parte inferior de vista Upload (con Metamask abierto)

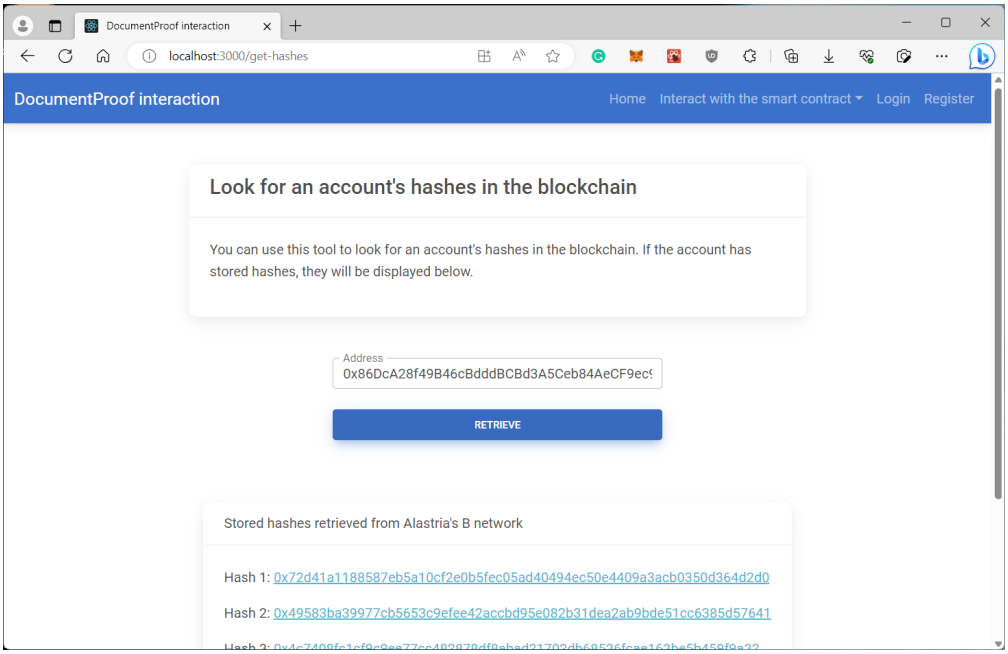


Figura 6.11: Vista de Retrieve Hashes

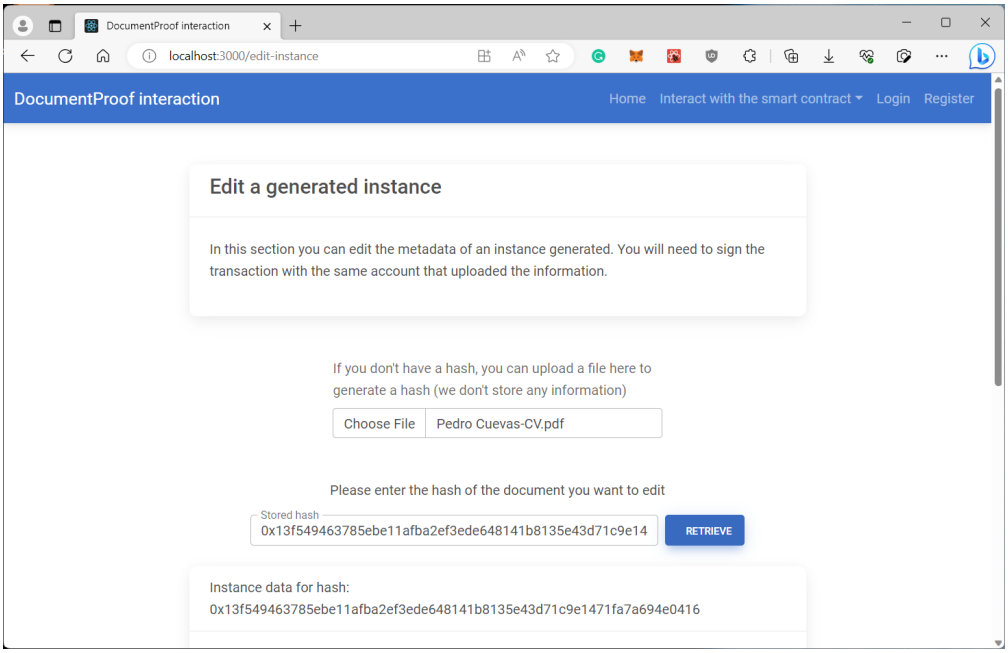


Figura 6.12: Parte superior de la vista de Edit Instance

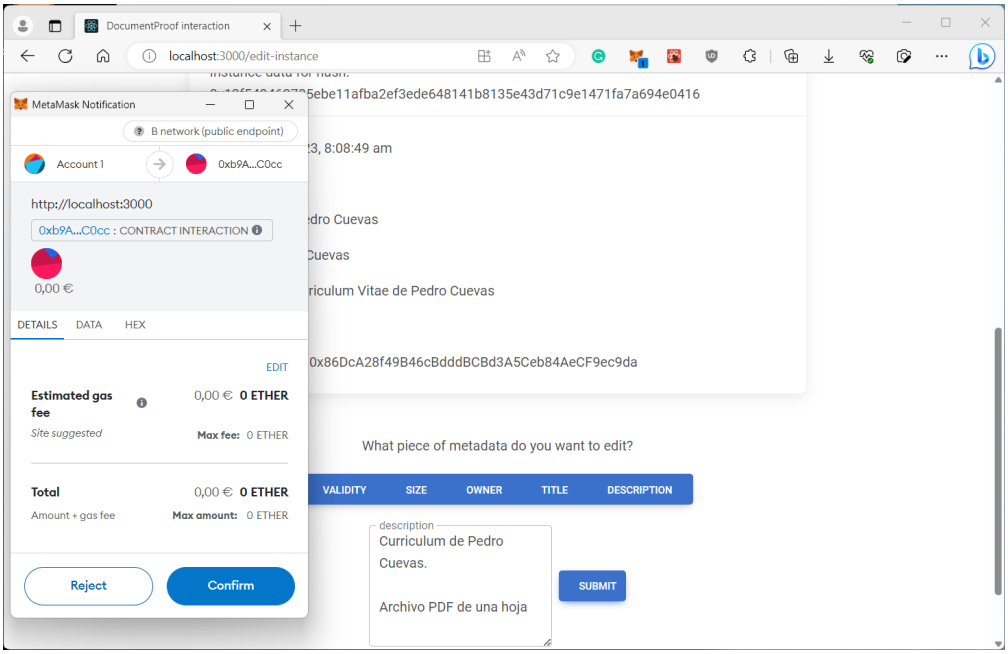


Figura 6.13: Parte inferior de la vista de Edit Instance (con Metamask abierto)

tance", para comprobar que el hash existe y mostrar la información almacenada en blockchain al usuario. En caso de éxito, el usuario verá la segunda parte, que es un grupo de botones donde el usuario elige qué información editar. Al pulsar el botón, se despliega el formulario correspondiente. El funcionamiento a partir de ese momento sería firmar la transacción con Metamask, y el feedback al usuario es el mismo que en las otras vistas. La única excepción es que, en caso de éxito, la información mostrada del smart contract se actualizará.

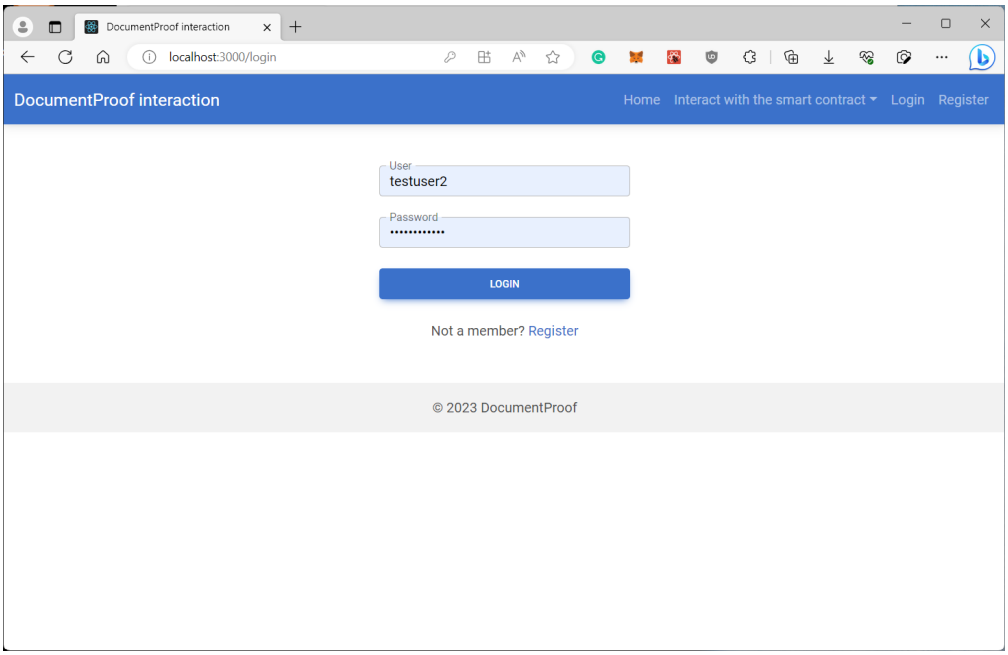


Figura 6.14: Login

La figura 6.14 muestra la vista de Login. Esta es la vista encargada de interactuar con el backend para obtener los tokens de autenticación, que son guardados en el frontend de React. Cuando un usuario se loguea con éxito, es redirigido a su vista de Dahsboard.

La figura 6.15 muestra la vista de Register. En caso de registrar el usuario con éxito, este será redirigo automáticamente a la vista de Login. Cabe destacar que, si bien el backend ya se encarga de comprobar que los campos tengan el formato buscado, el frontend también lo hace, para evitar llamadas innecesarias a la API.

La figura 6.16 muestra la vista de "Dashboard", que es desde donde el usuario registrado puede acceder a su información guardada en el servidor.

Los botones "save new hash", "edit" y "delete" despliegan Modals con formularios para que el usuario pueda teclear información, o simplemente confirmar la acción

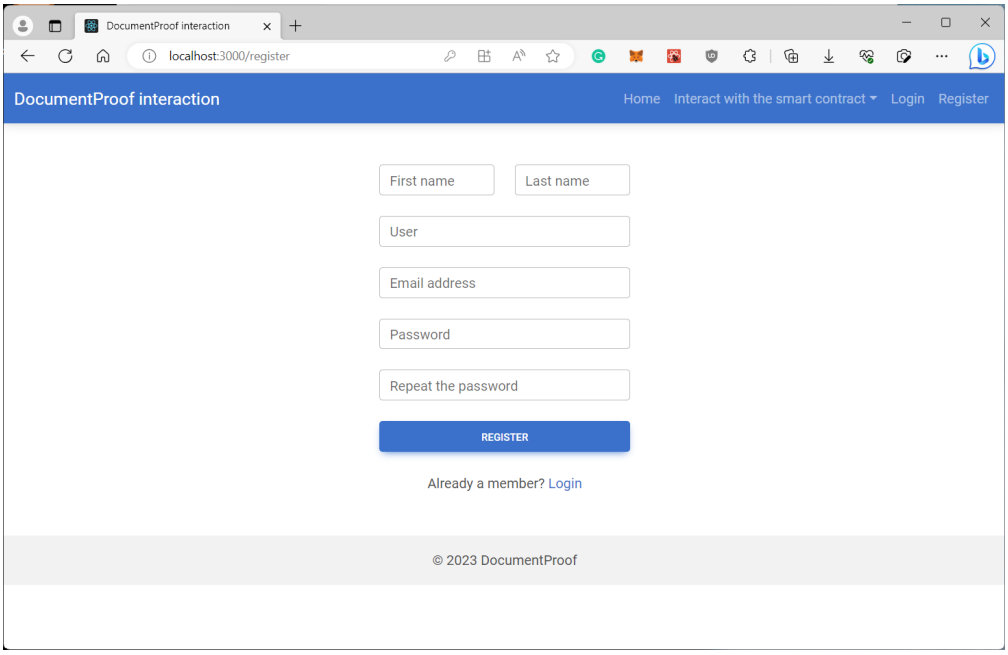


Figura 6.15: Register

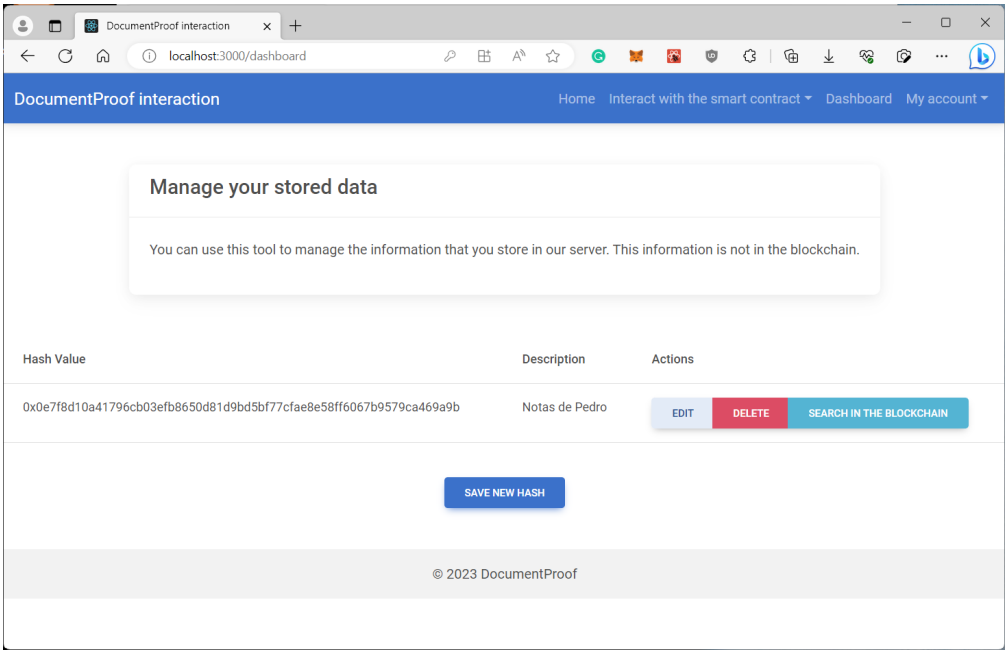


Figura 6.16: Dashboard

para el caso de delete. El botón de "search in the blockchain" redirecciona a la vista de Retrieve Instance con el hash ya escrito en el formulario.

Cuando un usuario está autenticado, el aspecto de la barra de navegación cambia: los botones de login y register desaparecen, y en su lugar aparecen el de dashboard y el menú desplegable "My account". Este menú presenta dos opciones: un botón de logout, y un botón para redirigir a la vista de "Account Settings".

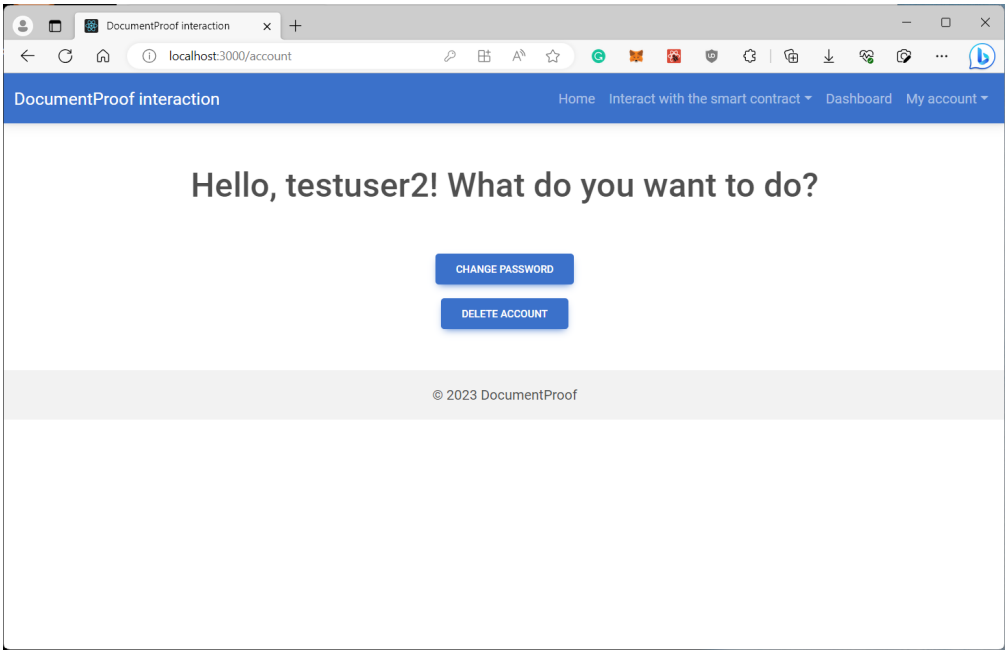


Figura 6.17: Account

Por último, la figura 6.17 muestra la vista de "Account Settings". Esta permite hacer dos acciones: cambiar la contraseña o eliminar la cuenta. En ambos casos, se volverá a deplegar un Modal para pedir al usuario la información necesario para llamar a la API, y pedir confirmación.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

A lo largo de este TFG, se han alcanzado satisfactoriamente los objetivos planteados inicialmente, conuna excepción:

- Se ha desarrollado e implementado un smart contract funcional que logra certificar credenciales académicas en la blockchain. Este contrato es capaz de almacenar de forma segura un hash de un documento y sus metadatos, ofreciendo transparencia, trazabilidad, y un sistema de verificación robusto basado en las ventajas de la Red B de Alastria.
- Se ha construido un frontend básico pero eficaz que permite al usuario una interacción intuitiva con el smart contract, facilitando la incorporación de nuevos documentos y su verificación.
- Adicionalmente, se ha construido un backend que permite generar y administrar usuarios. Además, los usuarios autenticados pueden guardar y editar información relacionada con hashes. Este backend cuenta con una REST API para poder interactuar co él.
- Se ha desarrollado un plugin de Moodle que permite a los usuarios interactuar con el smart contract desde Moodle, integrándolo con el sistema de archivos de Moodle.

El único objetivo que no se ha logrado ha sido el de integrar el plugin en el Moodle de la Universidad Pontificia Comillas. Esto se ha debido a que se trataba de un objetivo secundario, por lo que no ha sido priorizado a lo largo de la consecución del trabajo.

Asimismo, cabe añadir que tampoco se ha cumplido con los plazos establecidos en los objetivos del trabajo, pues una serie de motivos personales han retrasado la entrega de este TFG hasta agosto de 2023.

7.2. Trabajos futuros

A pesar de los logros conseguidos, hay espacio para mejoras y expansiones. A continuación, se detallan algunas propuestas para trabajos futuros:

- **Despliegue del plugin en el Moodle de la Universidad Pontificia Comillas**: Testear y mejorar el plugin, de manera que se pueda implementar en la universidad para aquellos profesores que quieran hacer uso del mismo.
- **Expansión del smart contract**: Ampliar las funcionalidades del contrato inteligente para incluir características adicionales, como la revocación de credenciales o la incorporación de firmas digitales de la institución.
- **Interfaz de usuario mejorada**: Aunque el frontend actual es funcional, se podría trabajar en una interfaz más amigable, responsiva y adaptada a diferentes dispositivos.
- **Seguridad y privacidad**: Investigar y aplicar medidas adicionales para garantizar la privacidad y protección de información sensible en la aplicación web
- **Integración con otras plataformas**: Expandir el sistema para ser compatible con otras plataformas educativas populares.

Finalmente, este TFG ha sido una oportunidad para explorar y aplicar tecnologías emergentes en un contexto educativo, mostrando su potencial y las amplias posibilidades que ofrecen para el futuro de la educación superior.

Apéndice A

Alineación con los Objetivos de Desarrollo Sostenible

El TFG se alinearía con los siguientes ODS:

Educación de Calidad (4)

Los sistemas de acreditaciones basados en blockchain tienen el potencial de dar el servicio de acreditaciones académicas seguras a bajo coste, por lo que permite dar a los colegios de bajos recursos una herramienta adicional para garantizar una mayor igualdad de oportunidades para sus alumnos.

Igualdad de Género (5)

La proporción de hombres que mienten en su CV es significativamente mayor que la de mujeres [1]. Por lo tanto, una herramienta dedicada a atajar ese problema tendría un claro impacto de género.

Trabajo Decente y Crecimiento Económico (8)

Mejorar los sistemas de reclutamiento de personal garantiza un acceso más meritocrático a las oportunidades. A su vez, esto permite que los individuos mejor

cualificados accedan a los mejores trabajos, aumentando su productividad y mejorando el desempeño económico de las instituciones para las que trabajen.

Reducción de las Desigualdades (10)

Un sistema de acreditaciones más transparente y barato permite que los individuos de entornos desfavorecidos o instituciones desconocidas puedan demostrar de manera más fiable sus capacidades, garantizando una mayor igualdad de oportunidades.

Paz, Justicia e Instituciones Sólidas (16)

En multitud de ocasiones, políticos se ven envueltos en escándalos por haber mentido en su CV o haber cometido alguna forma de fraude académico. La línea de trabajo de este TFG puede ayudar a desincentivar este tipo de comportamiento.

Bibliografía

[1] Resume Builder. 1 in 3 Americans Admit to Lying on Resume. Sep. de 2021. URL: https://www.resumebuilder.com/1-in-3-americans-admit-to-lying-on-resume/.

[2] Mario Becedas. Verdades y mentiras: ¿qué currículum tienen los políticos españoles? [Online; accessed 20-August-2022]. URL: https://www.eleconomista.es/noticias/noticias/8267120/04/17/Verdades-y-mentiras-que-curriculum-tienen-los-politicos-espanoles.html.

[3] Satoshi Nakamoto. «Bitcoin: A Peer-to-Peer Electronic Cash System». En: (2008). URL: https://bitcoin.org/bitcoin.pdf.

[4] Xiang Fu, Huaimin Wang y Peichang Shi. «A survey of Blockchain consensus algorithms: mechanism, design and applications». En: Science China Information Sciences 64.2 (nov. de 2020), pág. 121101. ISSN: 1869-1919. DOI: 10.1007/s11432-019-2790-1. URL: https://doi.org/10.1007/s11432-019-2790-1.

[5] Bruno Rodrigues et al. «A technology-driven overview on blockchain-based academic certificate handling». En: Blockchain Technology Applications in Education (2020), págs. 197-223. DOI: 10.4018/978-1-5225-9478-9.ch010.

[6] Vitalik Buterin. «Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform». En: (2014). URL: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[7] Rajeev Sobti y Geetha Ganesan. «Cryptographic Hash Functions: A Review». En: International Journal of Computer Science Issues, ISSN (Online): 1694-0814 Vol 9 (mar. de 2012), págs. 461-479.

[8] Jayakanth Srinivasan. Notes on hashing - MIT. [Online; accessed 20-August-2022]. URL: https://web.mit.edu/16.070/www/lecture/hashing.pdf.

[9] Home - Truffle Suite. [Online; accessed 20-August-2022]. URL: https://trufflesuite.com/.

[10] Django Overview. [Online; accessed 20-August-2022]. URL: https://www.djangoproject.com/start/overview/.

[11] SQlite, advantages and disadvantages. [Online; accessed 20-August-2022]. URL: https://www.javatpoint.com/sqlite-advantages-and-disadvantages.

[12] What is React? [Online; accessed 20-August-2022]. URL: https://www.w3schools.com/whatis/whatis_react.asp.

[13] XAMPP Tutorial. [Online; accessed 20-August-2022]. URL: https://www.javatpoint.com/xampp.

[14] Consejo General del Notariado. Quién es el notario. [Online; accessed 20-August-2022]. URL: https://www.notariado.org/portal/qui%C3%A9n-es-el-notario/.

[15] Colegio Notarial de Madrid. Legalización y apostillas. [Online; accessed 20-August-2022]. URL: https://madrid.notariado.org/portal/legalizacion-y-apostillas.

[16] Real Casa de la Moneda. ¿Qué es la Firma Digital? [Online; accessed 20-August-2022]. URL: https://www.sede.fnmt.gob.es/preguntas-frecuentes/otras-preguntas/-/asset_publisher/1RphW9IeUoAH/content/1026-que-es-la-firma-digital-#:~:text=Una%20firma%20digital%20es%20un,que%20est%C3%A1%20o%20no%20cifrado..

[17] Panda Security. ¿Es la firma digital tan segura como pensamos? [Online; accessed 20-August-2022]. URL: https://www.pandasecurity.com/es/mediacenter/seguridad/firma-digital/.

[18] MIT Registrar's Office. Digital Diplomas. [Online; accessed 03-November-2022]. URL: https://web.archive.org/web/20221103153726/https://registrar.mit.edu/transcripts-records/diplomas/digital-diplomas.

[19] Rodelio Arenas y Proceso Fernandez. «CredenceLedger: A Permissioned Blockchain for Verifiable Academic Credentials». En: (2018), págs. 1-6.

[20] Alastria. ¿Qué somos? [Online; accessed 22-November-2022]. URL: https://alastria.io/que-es-alastria/.

[21] Open Canarias SL. Open Digital Registry. [Online; accessed 23-November-2022]. URL: https://opendr.opencanarias.com/.

[22] Hyperledger Besu. Configure IBFT 2.0 consensus. [Online; accessed 23-November-2022]. URL: https://besu.hyperledger.org/en/stable/private-networks/how-to/configure/consensus/ibft/#genesis-file.

[23] Tabla de Amortización Simplificada. [Online; accessed 20-August-2022]. URL: https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html.

[24] Sueldos para el puesto de Software Engineer en España. [Online; accessed 20-August-2022]. URL: https://www.glassdoor.es/Sueldos/software-engineer-sueldo-SRCH_KO0,17.htm.

[25] El coste de los trabajadores para tu empresa. [Online; accessed 20-August-2022]. URL: https://www.personio.es/glosario/coste-de-trabajador-para-empresa/.

[26] ¿Cuánto cuesta un hosting en 2023? [Online; accessed 20-August-2022]. URL: https://www.hostinger.es/tutoriales/cuanto-cuesta-un-hosting.