UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

ContractMe

Presentado por David Martínez Bahillo
en Universidad de Burgos — 18 de junio de 2024
Tutor: Sandra Rodriguez Arribas

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. David Martínez Bahillo, con DNI 71566321G, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «ContractMe».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 18 de junio de 2024

Vº. Bº. del Tutor:

Sandra Rodríguez Arribas

Resumen

Este proyecto se enfoca en la creación de una solución innovadora para la contratación laboral, aprovechando la tecnología *blockchain* y los contratos inteligentes. Se busca agilizar el proceso de contratación en sectores como la agricultura y servicios domésticos mediante la automatización de la gestión de contratos desde su inició hasta su final. El proyecto incorpora métodos de autenticación biométrica y escaneo de códigos QR, apoyándose en soluciones de identidad descentralizadas para proteger la privacidad de los usuarios.

El objetivo final es proporcionar una herramienta que promueva prácticas de empleo transparentes y legales, optimizando los procesos de contratación y verificación para empleadores y trabajadores garantizando la seguridad de todos los trámites.

Descriptores

Blockchain, contratos inteligentes, autenticación biométrica, aplicación móvil, procesamiento pagos.

Abstract

This project focuses on creating an innovative solution for labor recruitment, taking advantage of blockchain technology and smart contracts. The aim is to streamline the contracting process in sectors such as agriculture and domestic services by automating contract management from start to finish. The project incorporates biometric authentication methods and QR code scanning, relying on decentralized identity solutions to protect user privacy.

The ultimate goal is to provide a tool that promotes transparent and legal employment practices, optimizing the hiring and verification processes for employers and workers, guaranteeing the security of all procedures.

Keywords

Blockchain, smart contracts, biometric authentication, mobile app, payment processing.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

La transformación digital se ha convertido en un pilar fundamental para el desarrollo y la eficiencia de diversos sectores económicos, entre ellos el mercado laboral. Sin embargo, a pesar de los avances tecnológicos, ciertos sectores como la agricultura, los servicios domésticos y el trabajo *freelance* enfrentan retos significativos en la gestión eficiente de contrataciones y la verificación de identidad. Estos desafíos se ven agravados por la presencia de la economía sumergida, donde la falta de transparencia y la informalidad laboral no solo perjudican la economía global, sino que también vulneran muchos de los derechos de los trabajadores. En este contexto, se pretende implementar una solución innovadora que permita superar las barreras existentes, haciendo frente a la economía sumergida mediante el empleo de tecnologías disruptivas como *blockchain* y contratos inteligentes. Esta iniciativa busca promover la formalización de empleos y asegurar prácticas laborales justas, abordando directamente los problemas de falta de transparencia y seguridad en los procesos contractuales. Se destacará el impacto social y económico de reducir la economía sumergida, apoyándose en datos que evidencian su magnitud y las implicaciones positivas de una mayor formalización laboral.

La adopción de contratos inteligentes en una red *blockchain* permite una gestión contractual transparente, segura y automática, asegurando que los términos acordados se cumplan eficazmente sin intervención manual. Esta tecnología también proporciona un registro inmutable y verificable de las transacciones y acuerdos labores, combatiendo así prácticas ilegales y fomentando un entorno de trabajo más equitativo.

Este proyecto no solo aborda la necesidad de un sistema de contratación y verificación más eficiente y seguro sino que también se anticipa a las demandas de un mercado laboral en constante evolución, ofreciendo una

herramienta adaptable y escalable. Con una interfaz de usuario diseñada para la simplicidad y la accesibilidad en dispositivos móviles, se busca fomentar prácticas de empleo legales y transparentes, contribuyendo a la creación de un entorno laboral más justo y seguro para todos. La inclusión de tecnologías de autenticación biométrica refuerza este objetivo, asegurando la identidad de los trabajadores de manera segura y privada.

Por tanto este proyecto no solo propone una solución práctica que beneficia tanto a trabajadores como empleadores, sino que también tiene el potencial de impactar positivamente en la economía global, marcando un paso adelante hacia la erradicación de la economía sumergida y el establecimiento de prácticas laborales más justas y transparentes a nivel mundial.

2. Objetivos del proyecto

En esta sección se muestran las metas que se pretenden alcanzar con el desarrollo del proyecto.

Objetivos técnicos

1. **Seguir estándares en el desarrollo de contratos inteligentes**: Alineándose con las mejores prácticas de la comunidad Ethereum y Solidity.

2. **Adoptar metodologías ágiles**: familiarizándose con el flujo de trabajo de Scrum para favorecer un desarrollo iterativo y eficiente, facilitando la adaptación a cambios y mejora continua del proyecto.

3. **Documentar del proyecto**: Utilizando herramientas visuales para explicar la arquitectura y la interacción entre sus componentes, incluyendo guías para el uso de herramientas y procedimientos estándar del proyecto.

4. **Mantener un alto estándar de calidad en el código**: Empleando herramientas de revisión de código.

5. **Crear una solución eficiente**: En términos de eficiencia.

6. **Planear un plan de implementeción y escalabilidad**

Objetivos del Software

1. **Integrar Ethereum en el proyecto**: Asegurando una implementación efectiva para el despliegue de contratos inteligentes
2. **Desarrollar contratos inteligentes con Solidity**: Asegurando un código eficiente y adaptado a las necesidades específicas de la contratación laboral.
3. **Implementar tecnologías de autenticación biométrica**: Utilizando métodos como la huella dactilar o el reconocimiento facial para la identificación del usuario.
4. **Implementar códigos QR**: Agilizando los procesos administrativos y operativos además de aplicar una capa adicional de seguridad.
5. **Recoger datos legales del usuario**: Integrando una base de datos donde se recoja datos personales del usuario para acreditarlo legalmente.
6. **Desarrollar una interfaz de usuario intuitiva y accesible**: Priorizando la simplicidad y la usabilidad para garantizar una plataforma fácil de utilizar para todos los usuarios.

3. Conceptos teóricos

En esta sección se sintetizarán algunos conceptos teóricos relevantes para la correcta compresión del proyecto.

3.1. Introducción a la *blockchain*

La *blockchain* proporciona un registro inmutable y en tiempo real de transacciones, emergió como una solución al problema de doble gasto [**?**] en transacciones digitales, un desafío que había eludido a los criptógrafos durante décadas, el cual consiste en la posibilidad de gastar una misma moneda digital más de una vez. Esto ocurre debido a la falsificación o duplicación de archivos digitales que representan la moneda. En 2008 una persona o grupo anónimo, bajo el seudónimo de Satoshi Nakamoto, publicó un documento técnico para crear una moneda digital para contabilizar y transferir valor. Así nació una tecnología que se fundamenta en una base de datos descentralizada compuesta por bloques de información replicados y sincronizados en múltiples ordenadores. Esta premisa hizo que en enero de 2009 entrara en funcionamiento la primera red basada en el protocolo Bitcoin [**?**]. La *blockchain* demostró su capacidad para contabilizar y transferir valor de manera segura sin la necesidad de intermediarios, transformando el sector financiero y encontrando aplicaciones en una variedad de industrias.

3.2. Tecnología subyacente

Nodo

Los nodos en *blockchain* son dispositivos, generalmente computadoras, que participan en una red *blockchain*. Estos dispositivos ejecutan el software del protocolo *blockchain*, lo que les permite ayudar a validar transacciones y mantener la seguridad de la red [**?**].

Para lograr esto, los nodos se comunican entre sí por medio de protocolos *peer-to-peer*(P2P). En un sistema P2P, cada nodo se conecta directamente a otros nodos sin necesidad de intermediarios. Cada nodo actúa tanto como cliente como servidor, compartiendo la carga de procesamiento de datos y la transmisión de información. Esta arquitectura (ver imagen 3.2) permite desarrollar un red robusta y descentralizada donde cada nodo contribuye al mantenimiento y seguridad de la red. Siendo extremadamente difícil censurar o bloquear el acceso a la red, ya que no hay un punto central de control.

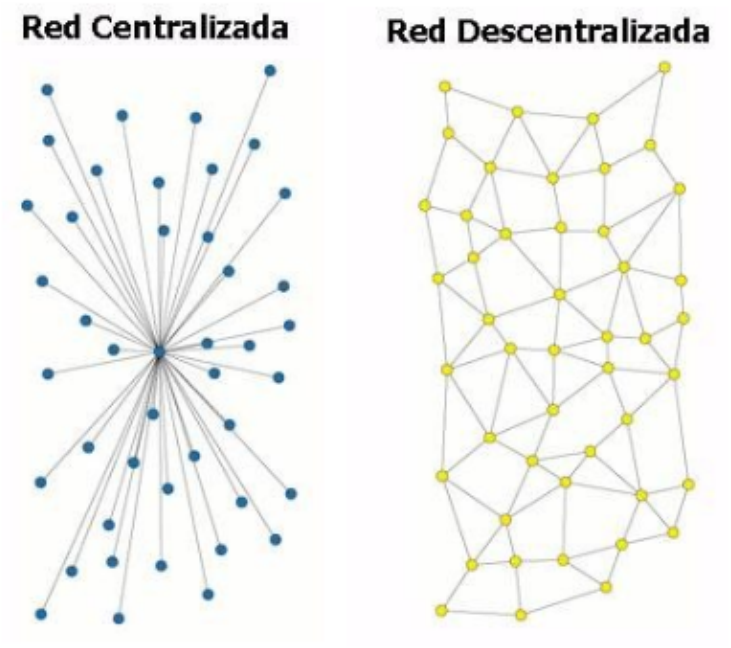

Figura 3.1: Diferencias entre una red centralidad y descentralizada [**?**].

Resulta fundamental comprender los distintos tipos de nodos que conviven en una red y el papel específico que cada uno desempeña. A continuación,

se enumerarán los tipos de nodos, destacando sus características y funciones únicas [**?**].

- **Nodos completos:** Estos nodos albergan una copia completa del libro mayor de la *blockchain*. Al contener el registro completo de todas las transacciones, los nodos pueden verificar de forma independiente cualquier transacción sin necesidad de recurrir a información externa.
- **Nodos ligeros:** Han sido diseñados para dispositivos con recursos limitados, los nodos ligeros no almacenan el registro completo, sino que confían en los nodos completos para obtener dicha información.
- **Nodos mineros:** Son utilizados en las redes que utilizan el algoritmo de consenso Proof of Work(PoW), los nodos mineros compiten para agregar nuevos bloques a la *blockchain* para obtener una recompensa por ello.
- **Nodos completos podados:** Estos nodos almacenan una versión recortada del registro, eliminando datos antiguos para ahorrar espacio, pero a diferencia de los nodos ligeros estos pueden seguir verificando de forma independiente cualquier transacción.
- **Nodos completos de archivo:** Almacenan todo el libro mayor del *blockchain*, desde el principio de los tiempos. Los nodos completos de archivo son la única fuente valiosa y fiable para verificar los datos de transacciones anteriores en la historia de una *blockchain*, ya que no están afectados por el límite de tiempo o almacenamiento. A diferencia de los nodos completos, los nodos de archivo vas más allá al almacenar cada cambio de estado en la *blockchain*.
- **Nodos de autoridad:** Utilizados en *blockchains* con mecanismos de consenso como Proof of Authority(PoA), estos nodos son operados por entidades verificadas y de confianza dentro de la red. A diferencia de PoW, tienen el poder de validar bloques sin necesidad de competir entre ellos.
- **Nodos maestros:** Ofrecen funcionalidades adicionales como la ejecución de contratos inteligentes. Requieren de una garantía o *stake* para operar y suelen recibir incentivos por ofrecer servicios especializados.
- **Nodos de estaca:** Utilizados en *blockchains* que funcionan con el algoritmo de consenso Proof of Stake(PoS) donde los nodos participan en la validación de bloques apostando una cierta cantidad de criptomonedas como garantía para operar.

- **Nodos *Lightning*:** Específicos de las soluciones de segunda capa como *Lightning Network*, estos nodos facilitan transacciones rápidas y de bajo costo fuera de la cadena principal, ayudando a reducir la congestión.
- **Supernodos:** Son nodos con capacidades y responsabilidades adicionales, a menudo relacionadas con la gobernanza de la red, se crean bajo demanda para realizar tareas especializadas, como implementar cambios de protocolo o gestionar protocolos.

Para este proyecto se ha utilizado Ethereum la cual utiliza una gran variedad de nodos para satisfacer las necesidades específicas de su ecosistema, incluyendo nodos completos, nodos ligeros, y nodos de archivo. A partir de la evolución de Ethereum 2.0 y su progresiva migración se han introducido los nodos estaca, que poco ha poco van remplazando a los nodos mineros utilizados en Ethereum 1.0.

Libro mayor digital

La tecnología *blockchain* funciona como un libro mayor digital (DTL) que registra transacciones en múltiples nodos de manera que cada registro es inalterable e irreversible. Este registro se organiza en bloques de datos que están interconectados de manera cronológica formando una cadena [**?**]. Una vez un bloque es añadido a la cadena, se distribuye a todos los nodos de la red, actualizando así el libro mayor en cada nodo. Esta distribución asegura una redundancia que la hace segura contra la manipulación, ya que alterar un registro requeriría cambiar el bloque correspondiente y todos los bloques posteriores en la mayoría de los nodos de la red, una tarea casi imposible debido a la demanda de cómputo [**?**]. Una representación gráfica se puede observar en la imagen 3.2.

Encadenamiento de bloques

Cada bloque contiene un numero determinado de transacciones y está formado por tres elementos principales; los datos de las transacciones, el *hash* del bloque anterior en la cadena y su propio *hash* único, generado a partir de la información contenida en el bloque. Los bloques se enlazan mediante el *hash* formado por los datos del bloque anterior. El algoritmo *hash* más usado en la *blockchain* es SHA-256 [**?**] desarrollado por la Agencia de Seguridad Nacional (NAS) en el año 1997. Es conocido por ser lento en comparación con otras funciones *hash*, pero a pesar de esto destaca por su seguridad por

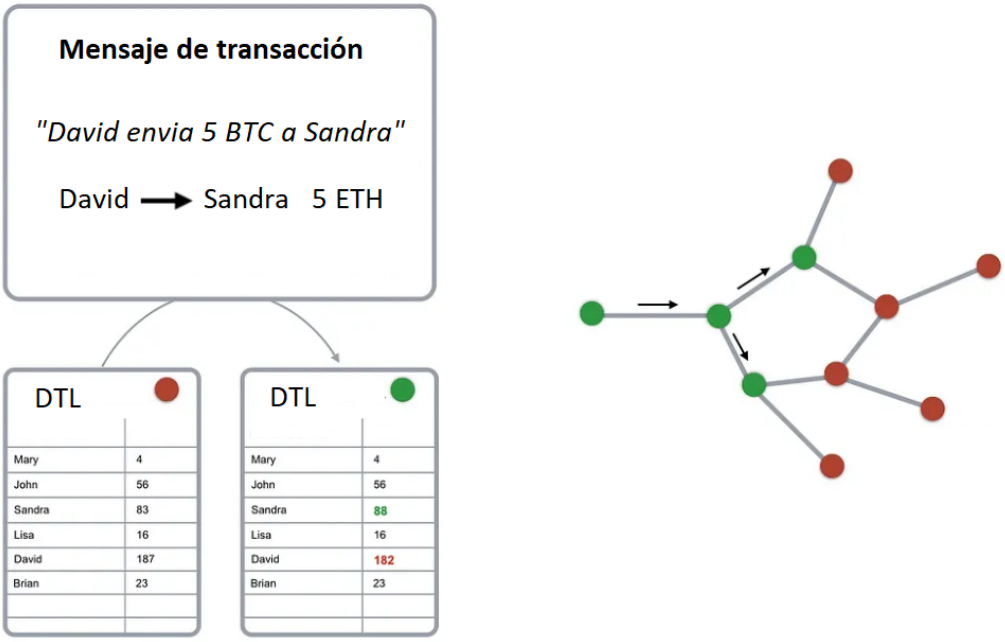

Figura 3.2: Proceso de actualización del libro mayor [**?**].

lo que lo hace adecuado para aplicaciones financieras. El *hash* consiste en una función criptográfica que produce una salida de longitud fija a partir de una entrada de longitud variable. Cualquier cambio en el contenido de un bloque anterior requeriría recalcular todos los *hashes* subsiguientes, lo cual es computacionalmente costoso y prácticamente imposible de realizar sin ser detectado. Ver imagen 3.2.

Wallet y transacciones

Para iniciar una transacción en la *blockchain* se debe de emplear una *wallet* o monedero electrónico, este es un *software* que permite almacenar y intercambiar activos digitales. Este monedero genera una y almacena un par de claves criptográficas, una clave pública que actúa como una dirección a la cual otros pueden enviar activos, y una clave privada, que se utiliza para firmar digitalmente las transacciones, asegurando que solo el propietario de la clave privada pueda autorizar la transferencia de sus activos.

Por tanto, cuando se desea realizar una transacción, el monedero electrónico firma la transacción utilizando su clave privada. Esta firma digital, generada a través de algoritmos de criptografía asimétrica como el RSA, es esencialmente un *hash* criptográfico de la transacción encriptado con la

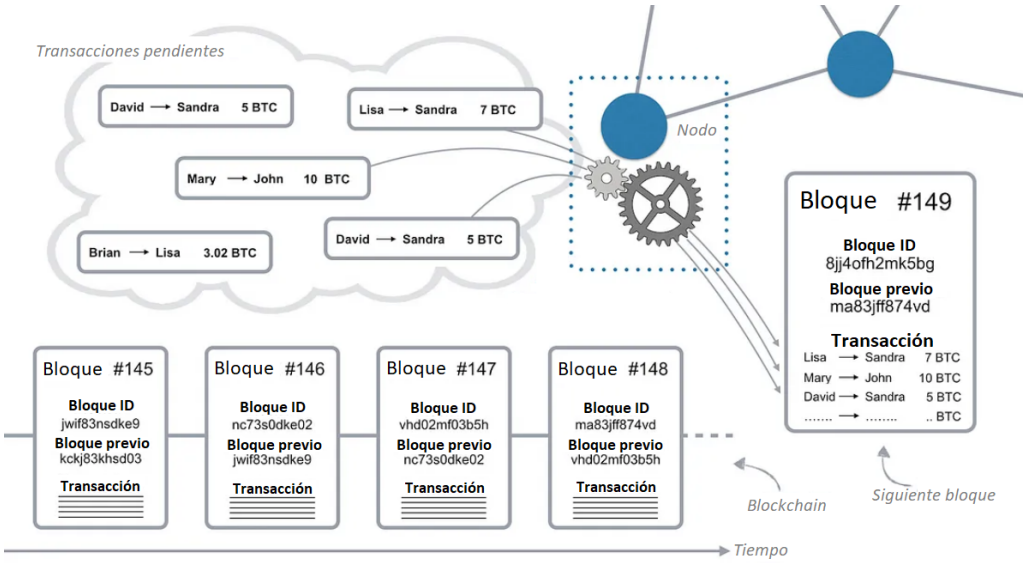

Figura 3.3: Encadenamiento de bloques en la *blockchain* [**?**].

clave privada del monedero. Seguidamente, los nodos de la red al recibir la transacción, emplean la clave pública del firmante para descifrar la firma digital. Este proceso no solo autentifica que la transacción fue creada por el poseedor de la clave privada correspondiente, sino que también asegura que la transacción no haya sido modificada, ya que cualquier cambio en los datos de la transacción resultaría en una discrepancia al verificar la firma con la clave pública. Ver imagen 3.2.

A diferencia de los sistemas financieros tradicionales, la *blockchain* no registra los saldos de las cuentas de manera directa. En su lugar, mantiene un registro detallado de todas las transacciones que han sido verificadas y aprobadas en la red. Por tanto, para determinar el saldo de una *wallet*, es necesario analizar y verificar todas las transacciones asociadas a ella desde la creación de la red. Este enfoque asegura que la información sea transparente y auditada constantemente por todos los nodos de la red, lo que refuerza la seguridad y la integridad del sistema.

Por tanto si un usuario quiere generar una transacción para enviar un activo, este debe generar una solicitud que incluya unas referencias llamadas *inputs*, a transacciones entrantes previas que sumen la cantidad deseada. Los nodos de la red verifican estos inputs para asegurarse que no hayan sido gastados previamente. Por ende, una vez los *inputs* hayan sido referenciados,

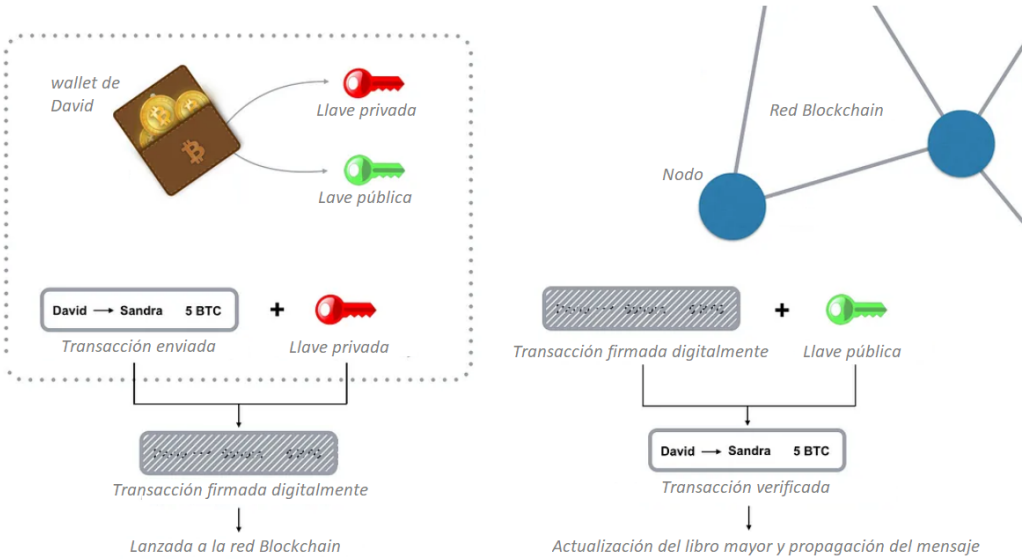

Figura 3.4: Transacción encriptada con firma digital [**?**].

estos son invalidados para transacciones futuras, evitando el doble gasto de activos digitales.

Consenso

El mecanismo de consenso en una *blockchain* es fundamental para mantener la integridad y la seguridad de la red. Este proceso permite que todos los nodos de la red se pongan de acuerdo sobre el estado actual del libro mayor digital, lo que significa que cualquier cambio en el *blockchain* debe ser validado y aceptado por más del 50 % de los nodos de la red.

Actualmente existen dos categorías amplias de protocolos de consenso, los de finalidad probabilística, representados fundamentalmente por **Proof of work** (PoW), implica que la confirmación de una transacción se vuelve más segura a medida que se van confirmando bloques sucesivos. Por otro lado, los protocolos con finalidad absoluta, como **Proof of Stake** (PoS) aseguran la finalización de las transacciones de manera definitiva una vez se agregan a la *blockchain*.

El protocolo Proof of Work se destaca como el mecanismo de consenso más empleado en las *blockchains*. En PoW, los participantes de la red, conocidos como mineros, compiten para resolver un problema criptográfico complejo, el cual requiere un considerable poder computacional. Este problema implica

encontrar un valor llamado *nonce* que cuando se combina con los datos del bloque y se procesa a través de una función *hash* produce un resultado que cumple con una criterio especifico, generalmente que contenga un cierto número de ceros al principio del *hash*. Este proceso, conocido como minería, garantiza que alterar un bloque ya minado sea computacionalmente inviable, proporcionando así seguridad e inmutabilidad a la cadena de bloques. A su vez, la dificultad de este problema se ajusta periódicamente para mantener un tiempo objetivo entre la creación de bloques consecutivos, asegurando la estabilidad y previsibilidad de la generación de nuevos bloques. La labor de los mineros en la *blockchain* no es altruista, este poder de cómputo es recompensado de tal manera que el primer minero en resolver el problema es recompensado con una cantidad fijada de criptomonedas [**?**].

A pesar de su amplia adopción y probada seguridad, La minería de criptomonedas requiere de grandes granjas de minado que conllevan un elevado consumo energético teniendo un gran impacto ambiental. Este aspecto ha impulsado la búsqueda de alternativas más eficientes y sostenibles como el Proof of Stake (PoS) que reduce el consumo eléctrico. Este protocolo se basa en seleccionar a los validadores en proporción a la cantidad de criptomonedas que poseen y están dispuestos a bloquear como garantía. La adopción de PoS por parte de proyectos líderes como Ethereum, con su transición a Ethereum 2.0, marca un hito importante y podría incentivar a otras criptomonedas a seguir un camino similar.

Hay que destacar que estos mecanismos de consenso únicamente son seguros en redes de una cierta magnitud ya que en contra pueden ser victimas del ataque del 51 %. El ataque del 51 % es una vulnerabilidad en la que un único minero o grupo de mineros controla más del 50 % del poder de cómputo de la red. De tal forma que les permite influir en las confirmaciones de las transacciones, permitiendo de esta forma el fenómeno del doble gasto. Es por esto que los mecanismos de consenso únicamente son efectivos en redes grandes, donde el tamaño de la *blockchain* es equivalente a la inversión necesaria para logar un control del 51 %.

3.3. Tipos de *blockchain*

Existen diferentes tipos de redes, cada una diseñada para satisfacer unas necesidades en cuanto a la privacidad, gobernanza y accesibilidad. Se pueden clasificar en cuatro categorías (ver tabla 3.1) principales [**?**]:

- ***blockchains* públicas:** Son completamente abiertas y cualquier persona puede unirse. Bitcoin y Ethereum son buenos ejemplos de *blockchains* públicas, donde las transacciones y los datos son visibles para todos, manteniendo al mismo tiempo el anonimato de los usuarios. Han marcado el camino para un ecosistema de aplicaciones descentralizadas (dApps) y finanzas descentralizadas (DeFi).

- ***blockchains* semiprivadas:** Son operadas por una única entidad con la posibilidad de restringir el acceso, ofreciendo un equilibrio entre el control y la descentralización. A diferencia de las redes privadas, las semiprivadas pueden permitir la participación de partes externas bajo ciertas condiciones, manteniendo un nivel significativo de control sobre la red. Este tipo de redes las ofrecen empresas como IBM (Hyperledger Fabric) utilizada en sectores como la salud y la financiación, permite a las organizaciones configurar redes donde los datos se comparten solo con los actores autorizados, mejorando la eficiencia y la seguridad. Ofreciendo opciones personalizables para empresas, equilibrando la privacidad con la innovación [**?**].

- ***blockchains* privadas:** Son operadas por una única organización, permiten un control total sobre quién puede participar en la red. Estas redes limitan el principio de la descentralización, pero ofrecen una solución eficaz para entornos empresariales que necesitan privacidad y eficiencia en procesos internos. Por ejemplo, para este proyecto se ha utilizado la herramienta Ganache, la cual simula una red privada, siendo de gran utilidad en la fase de desarrollo y pruebas.

- **Consorcio:** Representan un equilibrio entre los modelos públicos y privados, siendo operadas por un grupo de organizaciones en lugar de una única entidad. Esta posibilidad permite compartir la responsabilidad del mantenimiento de la red entre varios participantes, lo que las hace adecuadas para colaboraciones interempresariales. Un ejemplo real sería R3 Corda, que facilita transacciones eficientes y seguras entre instituciones financieras, reduciendo costos y tiempos de procesamiento [**?**].

3.4. Web3 y DApps

Web3 emerge como una propuesta revolucionaria en la evolución de Internet, promoviendo una arquitectura descentralizada que contrasta con

Característica	Pública	Privada	Consorcio	Semiprivada
Acceso	Abierto a todos	Restringido	Restringido a organizaciones	Control selectivo
Descentralización	Completa	Mínima	Parcial	Variable
Mecanismo de Consenso	PoW, PoS	Permisionados	Permisionados, personalizados	Combinación
Transparencia	Total	Limitada	Limitada a miembros	Configurable
Privacidad	Baja	Alta	Moderada	Alta en privado
Velocidad y Escalabilidad	Variable	Alta	Moderada	Configurable
Casos de Uso	DApps	Registros internos	Cadena de suministro, DeFi	Compartimentos privados

Tabla 3.1: Resumen de Tipos de *blockchain* y sus Características.

las fases anteriores de la web. Desde los comienzos de Internet con Web 1.0, caracterizada por páginas estáticas y un flujo unidireccional de información, hasta la aparición de Web 2.0, que permitía a los usuarios interactuar con el contenido en línea y entre ellos. Sin embargo, estas etapas se han caracterizado por su centralización del poder y los datos en mano de unas pocas plataformas dominantes, lo que a menudo cuestiona preocupaciones sobre la privacidad, seguridad y monopolizción de la información [**?**].

En este contexto nace Web3 como una solución prometedora para abordar estas preocupaciones centrándose en descentralización. Web3 a diferencia de sus antecesoras no se limita a ser un medio para compartir y crear contenido sino que pretende redefinir las dinámicas de poder en el espacio digital mediante la implementación de tecnologías *blockchain*.

La aplicación de la *blockchain* más reconocida a nivel mundial ha sido *Bitcoin*. Esta criptomoneda ha demostrado las capacidades de la *blockchain*, permitiendo transacciones globales rápidas y seguras, caracterizadas por su alta liquidez, bajas comisiones y un nivel de anonimato que protege la privacidad del usuario [**?**].

Más allá de las criptomonedas, esta tecnología se destaca por su naturaleza descentralizada, donde cada individuo en la red tiene acceso a una copia del registro completo de transacciones, lo cual garantiza una transparencia sin precedentes. Siendo así que la *blockchain* cuenta con la capacidad para ofrecer la trazabilidad completa en las cadenas de suministro. Cada producto puede ser rastreado desde su origen hasta el consumidor final, asegurando la autenticidad y facilitando la detección de cualquier problema en el proceso [**?**].

La seguridad es otro de los pilares fundamentales de la *blockchain*, ya que la inmutabilidad del registro asegura que una vez la información ha sido añadida a la cadena, esta no podrá ser alterada, reforzando así la confianza en el sistema.

Desde el punto de vista operativo, la *blockchain* ofrece eficiencias significativas al eliminar los intermediarios, reduciendo tanto los tiempos de procesamiento como los costos asociados, a parte de minimizar las posibilidades de error. Este aspecto es crucial en sectores como el financiero, donde los contratos inteligentes facilitan la ejecución automatizada y segura de acuerdos sin la necesidad de intermediarios, redefiniendo las prácticas comerciales y financieras en la era digital.

Dentro de este entorno de Web3, las aplicaciones descentralizadas (ver imagen 3.4) se presentan como un componente esencial, ofreciendo una alternativa a las aplicaciones centralizadas tradicionales. Aunque el concepto de DApp parece moderno, sus raíces se remontan más de 20 años. Las primeras aplicaciones en este ámbito fueron las aplicaciones de redes P2P, siendo algunas tan conocidas como eMule o BitTorrent, las cuales democratizaron el acceso a la información al distribuirla a través de una red de nodos (ordenadores) en lugar de centralizarla en servidores únicos [**?**].

El mercado de las DApps ha experimentado un crecimiento impresionante en los últimos años, escalando de un valor de mercado de 10.5 mil millones de dolares en 2019 a más de 25 mil millones en 2022, Se espera que este mercado mantenga un crecimiento acelerado, esperándose alcanzar una capitalización de 368 mil millones en 2027. Estas cifras subrayan la creciente importancia y potencial económico de las DApps en el ecosistema digital [**?**].

Hoy en día las DApps adquieren una nueva dimensión, ejecutándose en redes descentralizadas y apoyándose en la tecnología de contratos inteligentes para automatizar procesos y garantizar la ejecución de acuerdos sin intermediarios. Las Dapps destacan por su libertad y soberanía digital de los usuarios, ya que la ausencia de un punto central de control hace prácticamente imposible que se impongan restricciones arbitrarias por parte

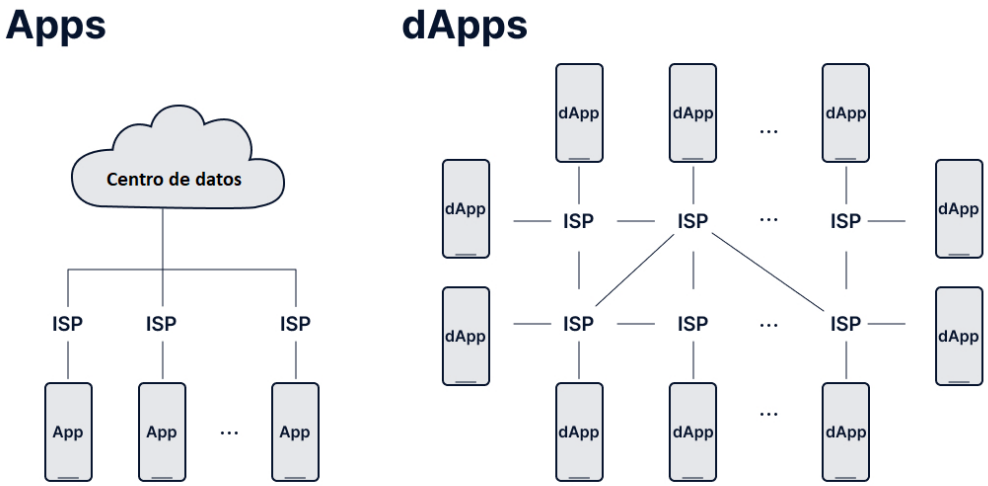

Figura 3.5: Estructura de las DApps [**?**].

de entidades externas. A su vez, esta distribución de los datos a través de la red dificulta los ataques y manipulaciones. Adicionalmente, el carácter de código abierto de muchas DApps fomenta la continua revisión por parte de la comunidad e incrementa la seguridad y la confianza en estas aplicaciones.

Las DApps se pueden clasificar en tres niveles distintos:

1. **primer nivel**: las de primer nivel se alojan en su propia *blockchain*.
2. **segundo nivel**: las de segundo nivel se alojan en *blockchains* ajenas.
3. **tercer nivel**: las de tercer nivel, las cuales dependen de Dapps de segundo nivel para funcionar.

Este proyecto se enmarca en las DApps de segundo nivel, ya que se opta por desarrollar la DApp sobre la *blockchain* de Ethereum, dada su amplia aceptación y su papel predominante en el ecosistema de las aplicaciones descentralizadas.

3.5. Ethereum

Ethereum actualmente es la segunda red más grande, solo por detrás de Bitcoin y aunque normalmente se les compara, Ethereum y Bitcoin son dos

proyectos totalmente distintos. Bitcoin introdujo una forma descentralizada de dinero electrónico, permitiendo realizar transferencias en una red segura y sin intermediarios. Ethereum por otro lado, lanzado en 2015 basándose en la base establecida por Bitcoin, expandió su funcionalidad.

La ambición de Ethereum no solo se limita a simplificar las transacciones financieras, su propósito es ampliar el alcance de las aplicaciones descentralizadas (Dapps) al proporcionar una infraestructura programable. A diferencia de Bitcoin, no cuenta con un suministro limitado de monedas para preservar su valor, además cuenta con tiempos de minado de bloques significativamente menores, por lo que puede ofrecer una experiencia más ágil y confirmaciones de transacciones más rápidas. Ethereum de esta manera se constituye como la columna vertebral de un Internet descentralizado(Web3) [**?**].

Ether

El Ether (ETH) es la criptomoneda que alimenta la red Ethereum, desempeña un papel fundamental tanto como activo digital como en la funcionaldiad de la red.

Similar a como ocurre con otras criptomonedas, ETH se puede enviar y recibir asegurando transacciones a cualquier parte del mundo y sin intermediarios. A su vez, cada operación que cambie el estado de la red, supone el pago de una pequeña tarifa en ETH. Desde una simple transferencia hasta la ejecución de complejos contratos inteligentes requieren computación y almacenamiento, siendo utilizado el ETH para pagar estas denominadas "Tarifas de gas". Finalmente como se introdujo el apartado mecanismos de consenso, Ethereum está implementando el mecanismo de consenso Proof of Stake(PoS) y el ETH constituye el activo que los participantes de la red pueden bloquear(stake) con el objetivo de dar seguridad a la red a cambio de recompensas en esta misma moneda.

Aunque ETH no tiene un suminsitro máximo fijo como Bitcoin, las políticas de emisión estan diseñadas para asegurar que el suminsitro de ETH crezca a un ritmo predecible y decreciente, lo que contribuye a la escasez y valor a largo plazo. El valor de ETH no se basa solo en su escasez digital como en el resto de criptomonedas, también adquiere un valor adicional al permitir a los usuario pagar las tarifas de gas y más recientemente ETH se ha vuelto válido para los usuario de aplicaciones financieras descentralizadas(DEFI) al poder usarse como garantía para prestamos criptográficos o como sistema de pago [**?**].

Ethereum Virtual Machine

La Ethereum Virtual Machine (EVM) constituye el núcleo de la red Ethereum, proporcionando un entorno de ejecución aislado y seguro para los contratos inteligentes y las aplicaciones descentralizadas. La EVM es una maquina virtual turing completa, que facilita la ejecución de código en el contexto de la *blockchain*. La EVM funciona como una máquina de pila, con una profundidad de 1024 items, cada item es una palabra de 256 bits, seleccionado para su utilización con la criptografía de 256 bits. Se ejecuta a través de códigos de operación, que realizan operaciones estándar de pila como XOR, AND, ADD, SUB, etc [**?**].

La EVM funciona mediante la ejecución de *bytecode*, un conjunto de intrucciones de bajo nivel que la máquina es capaz de interpretar y ejecutar directamente. El bytecode se obtiene de la compilación de contratos inteligentes escritos en lenguajes de alto nivel. La EVM es capaz de ejecutar código en un entorno completamente aislado, por lo que los contratos pueden oprar sin comprometer la seguridad de la red. Manejando el acceso a los recursos de los computadores y limitando sus acciones en un ambiente controlado. Esta seguridad se ve reforzada por su naturaleza descentralizada y distribuida, por lo que cualquier intento de manipulación mediante un código malicioso deberá enfrentarse al consenso de la red. De esta forma, Ethereum funciona como un ordenador mundial descentralizado de una general en una red entre pares.

La EVM cuenta con gran flexibilidad en cuanto a su capacidad para soportar una variedad de lenguajes de programación, facilitando así la adaptabilidad del sistema Ethereum. Además, la EVM permite que cualquier persona con acceso a Ethereum pueda desplegar sus propios contratos inteligentes, democratizando el acceso a la tecnología *blockchain* y creando una gran comunidad de desarrolladores [**?**].

Gas

El gas es una unidad que mide la cantidad de esfuerzo computacional requerida para ejecutar operaciones en la red Ethereum. La tarifa de gas es la cantidad de gas usado para hacer alguna operación, multiplicado por el coste unitario del gas. Es un mecanismo no solo de prevención contra ataques de *spam*, sino tambien que facilita una economía alrededor, estableciendo un sistema de compensación para los mineros que procesan y confirman las transacciones. El precio del gas suele expresarse en gwei, cada gwei equivale a $0,000000001$ ETH o 10^{-9} ETH.

Bloque	Total Gas	Incremento Tarifa (%)	Tarifa Base (gwei)
1	15 M	0 %	100 gwei
2	30 M	0 %	100 gwei
3	30 M	12,5 %	112,5 gwei
4	30 M	12,5 %	126,6 gwei
5	30 M	12,5 %	142,4 gwei
6	30 M	12,5 %	160,2 gwei
7	30 M	12,5 %	180,2 gwei
8	30 M	12,5 %	202,7 gwei

Tabla 3.2: Evolución de la Tarifa Base en Ethereum.

Desde 2021 con la actualización L̈ondonse introdujo el mecanismo EIP-1559, que hizo que el calculo de la tarifa de gas fuera más previsible, introduciendo los conceptos de tarifa base y tarifa prioritaria.

La tarifa base (ver tabla 3.2) indica la cantidad mínima de gas a pagar para que la transacción se considere como válida para tramitar. La tarifa base se ajusta dinámicamente bloque a bloque basándose en la ocupación del bloque anterior. El sistema, calcula cuánto gas se utilizó en total para todas las transacciones en el bloque anterior y lo compara con el tamaño de bloque previamente definido por el protocolo. Este tamaño de bloque previamente definido es una medida de cuánto gas se espera que consuman las transacciones en un bloque ideal, cada bloque en la red Ethereum tiene un tamaño esperado de 15 millones de gas. Si el bloque anterior consumió más gas del tamaño previamente definido, la tarifa base aumentará para el siguiente bloque hasta un máximo de 12,5 %. El crecimiento exponencial en el costo de las transacciones cuando los bloques están constantemente sobreocupados actúa como un freno contra la congestión y alcanzando un tamaño de equilibrio de 15 millones de gas de media por bloque , ya que solo las transacciones prioritarias de aquellos que estén dispuestos a pagar más serán ejecutadas.

Por otro lado, **la tarifa prioritaria** establece una propina que se añade a la tarifa base para que los validadores vean la transacción más atractiva para incluirla en el siguiente bloque, ya que es esta propina la que adquirirán en recompensa a su trabajo, a su vez la tarifa base es consumida y por lo tanto, eliminada. Por lo general, la tarifa base por si sola es insuficiente para que un validador se interese por ella, por lo que la elección de una transacción por parte del validador depende de la tarifa prioritaria, la cual

tiene que ajustar su precio en base al uso de la red en el momento de enviar la transacción.

Existe otro tipo de tarifa que se puede incluir de manera opcional en las transacciones, esta es la tarifa máxima, que es la que se ha usado para las transacciones dentro de este proyecto. Este parámetro opcional recoge la cantidad máxima que el usuario está dispuesto a pagar por la validación de una transacción, por lo que dependiendo de la importancia de la transacción se puede consentir pagar más a cambio de una mayor velocidad de validación. La tarifa total constituye la suma de la tarifa base y la tarifa prioritaria [**?**].

Faucet

Los *faucets* representan una herramienta diseñada para facilitar a los usuarios la obtención de pequeñas fracciones de criptomonedas a cambio de realizar actividades simples, como captcha o pequeñas tareas. El primer faucet de criptomonedas se remonta a 2010, creado por Gavin Andresen, quien entonces era el desarrollador principal de Bitcoin. Este faucet distribuyó 5 BTC a cada usuario que resolviera un *captcha* simple, totalizando una entrega de 19,715 BTC [**?**]. Este mecanismo pretendía expandir la educación y adopción del Bitcoin entre un público más amplio. A día de hoy los *faucet* no ofrecen recompensas tan elevadas debido a la apreciación en el valor de las criptomonedas pero siguen teniendo un papel vital en la atracción y educación de nuevos usuarios mediante una vía de acceso sencilla y de bajo riesgo.

Hoy en día este enfoque se ha expandido y los *faucets* se han popularizado entre las *testnets*. Estos *faucets* son esenciales para el desarrollo y la prueba de aplicaciones descentralizadas (dApps), facilitando un ambiente vital para la innovación y el perfeccionamiento de nuevos productos y servicios en la *blockchain*.

Contratos inteligentes

La idea fue conceptualizada por primera vez por Nick Szabo en 1993, visionando una nueva forma de establecer acuerdos digitales. Sin embargo, la falta de una plataforma adecuada mantuvo esta idea en teoría hasta la llegada de la *blockchain* con Bitcoin en 2009, y mas notablemente con Ethereum en 2014, que los contratos inteligentes se materializaron prácticamente gracias a la infraestructura que esta tecnología proporciona [**?**].

Un contrato inteligente es un código que ejecuta automáticamente los términos de un acuerdo entre partes. Los códigos, almacenados en la *block-*

chain, son ejecutados automáticamente, cuando se cumplen unas condiciones predefinidas, haciendo cumplir un acuerdo entre dos partes no confiables sin la necesidad de un tercero de confianza. Los contratos inteligentes utilizan la tecnología *blockchain* para almacenar reglas, ejecutar automáticamente acciones cuando se cumplen esas reglas y almacenar los resultados en la *blockchain*. Debido a su naturaleza inmutable y distribuida, ofrecen un nivel de seguridad y confianza superior al de los sistemas tradicionales. Así mismo, al eliminar los intermediarios, ofrecen una reducción de costos y una mayor rapidez en la ejecución de acuerdos.

Sin embargo, se enfrentan a desafíos en cuanto a cuestiones legales, la necesidad de recursos externos a la cadena de bloques, su naturaleza inmutable que dificulta la corrección de errores, problemas de escalabilidad y limitaciones del mecanismo de consenso. Las soluciones de Capa 2, como la Lightning Network y Ethereum Plasma, se diseñaron para abordar los desafíos de escalabilidad y eficiencia de las *blockchain* de Capa 1. Operan sobre la cadena principal para permitir transacciones más rápidas y con menores costos, manteniendo la seguridad y descentralización [**?**].

3.6. Tokenización

La tokenización [**?**] es el proceso de convertir la información delicada o activos del mundo real en representaciones digitales denominadas *tokens*, dentro del ecosistema *blockchain*. Este procedimiento juega un papel crucial en la protección de datos confidenciales al reemplazar la información original con un *token* único, el cual no tiene valor fuera de su contexto específico de uso.

La información sensible se almacena en la 'bóveda de tokenización' [**?**] una infraestructura de almacenamiento segura donde los datos originales se cifran y aíslan. El acceso a la bóveda es solo posible a través de rigurosos controles de seguridad y claves de descifrado específicas. A diferencia de los métodos de cifrado que utilizan un algoritmo matemático para transformar datos en un formato ilegible que puede ser revertido usando un clave concreta, la tokenizción no mantiene una relación algorítmica con los datos originales. En consecuencia, los *tokens* generados no pueden ser revertidos sin un acceso autorizado a la bóveda de tokenización, lo que proporciona una capa adicional de seguridad. Por lo tanto, mientras los datos originales se almacenan en una bóveda de *tokens* segura, los *tokens* se distribuyen en sistemas internos para su utilización diaria.

Un elemento importante de los *tokens* es que, fuera de la relación financiera específica para la que fueron creados, carecen totalmente de valor. Ya que una función de los mismos es representar un valor específico en una relación determinada. Esta característica los distingue de las criptomonedas y otros activos digitales que pueden tener un valor en el mercado abierto. En el ámbito de los pagos y transacciones, los *tokens* permiten a las organizaciones procesar transacciones y almacenar información de clientes sin exponer los datos críticos a riesgos de seguridad. La generación de un *token* se realiza mediante contratos inteligentes en la *blockchain*, que definen las reglas y la lógica para su emisión, transferencia y anulación. Los contratos inteligentes aseguran que el token sea único y esté vinculado de manera inmutable a los datos o activos correspondientes en la bóveda.

En el ecosistema *blockchain* existen diversos tipos de tokens diseñados para propósitos específicos [**?**]:

- **Tokens de Seguridad:** Representa inversiones digitales en activos reales como acciones o bonos, respaldado por activos tangibles y regulado por entidades gubernamentales.
- **Tokens de Gobernanza:** Permiten a los poseedores participar en la toma de decisiones dentro de una plataforma o protocolo, votando en cambios o propuestas.
- **Tokens de Utilidad:** Proporciona acceso a productos o servicios dentro de una plataforma *blockchain*, sin ser considerado un valor financiero.
- **Tokens Comunitarios:** Recompensan la participación en una comunidad, ofreciendo beneficios como acceso exclusivo o descuentos a los miembros activos.
- **Tokens Vinculados a valores:** Son digitales pero están respaldados por activos físicos como metales preciosos, permitiendo a los inversores negociar activos reales de manera digital.

Todos los tipos de tokens existentes se pueden clasificar en dos grandes grupos, los tokens fungibles y los tokens no fungibles.

Tokens fungibles

La definición de fungibilidad es esencial para entender los aspectos fundamentales de un token fungible. Tomando como referencia una definición

ofrecida por el Tesoro Público del Gobierno de España, la fungibilidad se describe como la 'Propiedad de un conjunto de valores que los hace plenamente equivalentes entre sí a efectos legales' [**?**]. La fungibilidad es un concepto que nos rodea en la vida cotidiana, siendo el dinero uno de los mejores ejemplos. Cuando se intercambia un billete de cinco euros por otro billete de cinco euros, se entiende que ambos tienen el mismo valor y son aceptados de la misma manera.

Este principio se puede extrapolar al mundo digital y a la *blockchain*. Los token fungibles actúan de forma similar al dinero físico, siendo indistinguibles y equivalentes entre unidades del mismo tipo. El ejemplo más conocido es Bitcoin, convirtiéndolo en una herramienta poderosa para las transacciones digitales.

Tokens no fungibles (NFT)

Los NFT han emergido como una innovación disruptiva en el ámbito del comercio electrónico, especialmente en el mundo del arte digital. A diferencia de los tokens fungibles, cada NFT es una certificación criptográfica que contiene información y códigos de identificación únicos que los hacen irremplazables e intercambiables. Esta característica los hace particularmente adecuados para representar activos digitales únicos y derechos de propiedad en el mundo digital. En el contexto de los contratos laborales, los NFT pueden ser utilizados para tokenizar y asegurar la autenticidad de contratos individuales, garantizando que los términos acordados sean únicos y vinculados inequívocamente a las partes involucradas [**?**].

ERC-721

ERC-721 es un estándar propuesto por el desarrollador Dieter Shirley a finales de 2017 que introdujo el concepto de tokens no fungibles en la red Ethereum [**?**]. Abriendo las puertas a una nueva dimensión de activos digitales únicos, a diferencia de los tokens fungibles basados en el estándar ERC-20.

La creación del estándar ERC-721 fue motivada por la creciente demanda de tokens digitales que pudieran representar de manera única activos individuales. La singularidad de los tokens ERC-721 les dota de gran utilidad en aplicaciones donde el ámbito de autenticidad y la propiedad exclusiva son cruciales. Su uso mas popular se enfoca en el mundo del arte, asegurando la autenticidad y unicidad de diferentes obras, aunque también ha tomado gran relevancia en el ámbito legal. Poniendo de ejemplo este proyecto, con el uso

del estándar ERC-721 se puede tokenizar y autenticar contratos laborales, asegurando la transparencia y la inmutabilidad de los términos acordados, verificando el cumplimiento de los acuerdos.

Este estándar cuenta con una serie de propiedades técnicas que lo hacen versátil. Algunas de estas propiedades son la asignación de un nombre, la definición de un balance de tokens dentro de una dirección y la implementación de funciones que permiten la transferencia segura de la propiedad.

4. Técnicas y herramientas

En esta sección se exponen las herramientas y bibliotecas que se han aprendido y utilizado a lo largo del desarrollo del proyecto. A lo largo del ciclo de vida del proyecto, el proyecto experimentó varios cambios, incluyendo la sustitución o eliminación de algunas bibliotecas. Sin embargo, es importante destacar todas estas herramientas, ya que cada una ha tenido un impacto significativo en el proyecto y han sido fundamentales en diferentes etapas del mismo.

4.1. Herramientas

Se muestra a continuación las herramientas usadas a lo largo del desarrollo del proyecto.

React Native

La elección del *framework* adecuado juega un papel crucial en el éxito de un proyecto. Dicha elección se complica aún más cuando se carece de experiencia previa en el desarrollo de aplicaciones móviles. Dentro del gran abanico de posibilidades React Native y Flutter emergen como grandes líderes en el sector gracias a sus grandes comunidades y la abundancia de recursos en línea. Por otro lado se pueden descartar directamente opciones como el entorno de desarrollo de iOS (por ejemplo, el uso de Swift y Cocoa Touch) debido a su exclusividad para aplicaciones para iOS debido a que el objetivo de este proyecto es desarrollar una aplicación para Android.

React Native se presenta como mi elección favorita. Este es un *framework* de código abierto creado por Facebook orientado a la creación de aplicacio-

nes nativas tanto en iOS como en Android. React Native está basado en JavaScript y React, una biblioteca de JavaScript destinada a la creación de interfaces de usuario. Fue lanzado en 2015 con el propósito de superar las limitaciones del desarrollo de aplicaciones móviles basado solo en HTML5, en las que simplemente se adapta aplicaciones web a un entorno móvil. React Native utiliza componentes nativos en lugar de WebViews para la interfaz de usuario, esto conlleva a que las aplicaciones se sientan y actúen como en una aplicación nativa. Uno de los elementos más importantes de este *framework* es el ´React Native Bridge' (ver imagen 4.1), el cual facilita la comunicación entre el código JavaScript y los elementos nativos del dispositivos. El ´puente' maneja de forma paralela dos flujos de trabajo, uno ejecuta la lógica de la aplicación en JavaScript y otro gestiona las operaciones de la interfaz de usuario nativa. Esto permite que las aplicaciones en React Native accedan a las características del dispositivo como la cámara o la ubicación, ofreciendo una experiencia fluida para el usuario gracias a características como el *hot reload*.

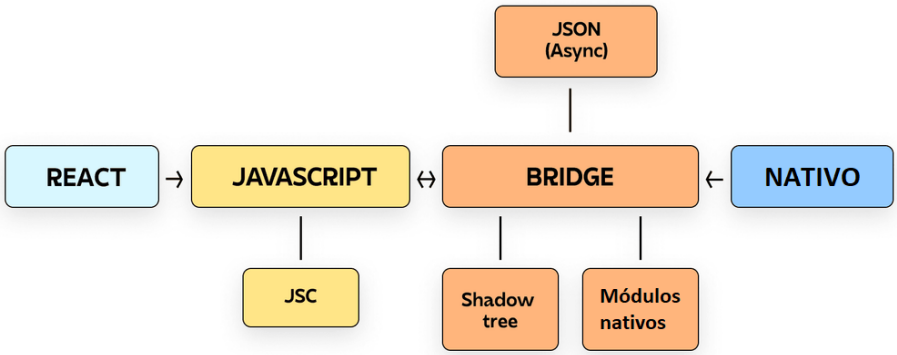

Figura 4.1: Flujo de trabajo y arquitectura de una aplicación React Native [**?**].

Mi preferencia de este *framework* sobre el resto radica en mi familiaridad previa con la programación web y JavaScript, lo cual reducirá la curva de aprendizaje en la transición hacia React Native en contraste con otros *framework* que podrían requerir el aprendizaje de nuevos lenguajes de programación o paradigmas de programación. Por otro lado, uno de los grandes motivos de este *framework* es su gran popularidad, el cual se traduce

en una gran riqueza de recursos disponibles, como bibliotecas, videotutoriales o foros, cruciales a la hora de enfrentar los desafíos que puedan surgir. Finalmente aunque para este proyecto no sea un requerimiento hay que tener en cuenta la gran versatilidad de React Native, el cual permite la creación de aplicaciones nativas tanto en Android como en iOS a partir de un único código base. Optimizando así el proceso de desarrollo permitiendo en un futuro poder dar cobertura a un mercado más amplio sin esfuerzos duplicados.

Como había nombrado anteriormente Flutter es otro *framework* que destaca sobre el resto y ofrece ciertas ventajas notables sobre React Native, como un rendimiento superior gracias al uso de su motor de renderizado propio y la gran capacidad de personalización de la interfaz de usuario. Aunque Flutter pudiera ser superior en algunos aspectos técnicos sigo decantándome por React Native debido a su gran comunidad y la familiaridad que tengo con las tecnologías web, priorizando así un aprendizaje más sencillo frente a posibles mejoras en el rendimiento. Por otro lado existen los *frameworks* Ionic y Xamarin que los he descartado debido a sus limitaciones en términos de acceso a funciones nativas y comunidades bastante inferiores comparado con React Native o Flutter. Kotlin era un opción con gran potencial para el desarrollo nativo de aplicaciones Android pero la descarté rápidamente por tener una curva de aprendizaje más pronunciada que su competencia, ya que representaría un obstáculo significativo en cuanto al tiempo de desarrollo sin garantizar beneficios proporcionales a dicho esfuerzo.

Respecto a Angular, si bien este *framework* ofrece un ecosistema robusto para el desarrollo de aplicaciones web dinámicas y complejas utilizando TypeScript, es importante mencionar que también es posible desarrollar aplicaciones móviles con Angular, ofreciendo una experiencia cercana a la nativa, aunque a través de un enfoque diferente al de las aplicaciones nativas desarrolladas con React Native. Angular no deja de ser una elección excelente, pero debido a la preferencia del proyecto de enfocarse en una aplicación móvil sin la necesidad de dar soporte de navegador, Angular puede no ser la opción más adecuada para el proyecto. Por lo tanto he descartado Angular buscando una solución más enfocada y optimizada para el desarrollo móvil, aprovechando las capacidades nativas de los dispositivos móviles.

Expo

Expo es un *framework* que simplifica el desarrollo de aplicaciones móviles con React Native, actuando como una capa de abstracción reduciendo las

barreras de entrada al desarrollo móvil. Consta de un entorno y herramientas listas para usar, eliminando la necesidad de configurar sistemas de construcción nativos complejos. Ofrece un conjunto de APIs disponibles a través de módulos que se pueden invocar directamente desde el código JavaScript las cuales cubren funcionalidades comunes del dispositivo, como cámara, geolocalización o notificaciones sin requerir de configuración adicional.

Expo ha sido fundamental en el proyecto para desarrollar la interfaz de usuario permitiendo probar y visualizar cambios en tiempo real en diferentes plataformas. Una de las funcionalidades significativas de Expo ha sido su capacidad para facilitar la ejecución y testeo de la aplicación directamente en dispositivos móviles personales sin necesidad de configurar emuladores. Esto se logra mediante el uso de la aplicación Expo Go en el teléfono móvil y escaneando un código QR generado por el entorno de desarrollo Expo. Expo ha tenido un papel fundamental en las fases de prueba y depuración, permitiendo probar la aplicación en un entorno real y ajustar la interfaz de usuario con un ciclo de retroalimentación casi instantáneo.

Firebase

Firebase es una plataforma de Google creada para el desarrollo de aplicaciones móviles y web. Cuenta con un amplio conjunto de servicios integrados que permiten crear aplicaciones robustas y de alta calidad sin necesidad de gestionar la infraestructura subyacente.

Ente el amplio abanico de servicios que ofrece, destacan los siguientes:

- **FireBase Authentication:** Provee diversos servicios de autenticación de usuarios, como inicio de sesión con correo electrónico, Google, FaceBook y Github entre otros. Facilitando la implementación de una amplia gama de servicios de autenticación sin necesidad de manejar detalles complicados de seguridad.

- **FireStore:** Ofrece soluciones de bases de datos en tiempo real y basadas en la nube. Esto permite sincronizar datos entre los usuarios en tiempo real, a parte, FireStore ofrece una estructura fácilmente escalable.

- **FireBase Cloud Messaging:** Sistema que permite enviar notificaciones a los usuarios de forma segura mediante correo y teléfono. Aunque para la versión gratuita este servicio es muy limitado.

- **FireBase Analytics:** Ofrece una herramienta poderosa para analizar el uso de la aplicación por parte de los usuarios. Permitiendo rastrear eventos específicos, obtener información demográfica de los usuarios, analizar la duración y frecuencia de las sesiones, y monitorear la retención y la ejecución de objetivos preestablecidos.
- **FireBase Cloud Functions:** Permite ejecutar código backend en respuesta a eventos desencadenados por alguno de los servicios de Firebase. Es de gran utilidad para ejecutar ciertas tareas en el lado del servidor, como procesamiento de pagos o el envió de correos electrónicos.

Es una herramienta de gran potencial que se integra perfectamente con React Native a través de módulos y SDKs. Proveyendo a los desarrolladores una variedad de herramientas preconstruidas y escalables a medida que la aplicación crece sin necesidad de ninguna intervención.

Solidity

Solidity es un lenguaje de programación de contratos orientado a objetos y de alto nivel diseñado específicamente para escribir *smart contracts* que se ejecutan en la Ethereum Virtual Machine (EVM). Consta de una sintaxis que recuerda a lenguajes como JavaScript, C++ o Python, lo que facilita su aprendizaje. Es un lenguaje de tipado estático, asegurando que el tipo de cada variable se define en tiempo de compilación y no cambia durante al ejecución del contrato, aumentando así la seguridad y ayudando a la detección de errores antes de la ejecución.

Una de las características más destacadas de Solidity es su soporte para la herencia, una característica común en la programación orientada a objetos. Esto permite que los *smart contracts* hereden propiedades y comportamientos de otros contratos, beneficiándose de la reutilización de código y la organización de la lógica del mismo. En mi caso ha sido de gran utilidad para implementar las funcionalidades del estándar ERC-721 utilizando la biblioteca OpenZeppelin, que ofrece un conjunto de contratos inteligentes auditados y aprobados por su comunidad.

Aunque Solidity no sea el único lenguaje de programación destinado a la creación de *smart contracts*, mi preferencia por este lenguaje se fundamente en dos aspectos claves. En primer lugar, mi familiaridad previa con lenguajes como JavaScript y Python hace que la transición a Solidity sea más intuitiva, a diferencia de Vyper cuya sintaxis presentan mayores diferencias. Por otro

lado, uno de los requisitos fundamentales de este proyecto era la capacidad del lenguaje para soportar herencia, una característica fundamental para implementar el estándar ERC-721, requisito que Vyper no incluye.

Remix

Remix es un entorno de desarrollo integrado (IDE) diseñado para el desarrollo de *smart contracts* escritos en Solidity. Proporciona una interfaz accesible y amigable para escribir, compilar, probar y desplegar *smart contracts* directamente desde el navegador, sin la necesidad de instalar software adicional. Remix también ofrece funcionalidades avanzadas como la compilación en tiempo real, el despliegue de contratos en diversas redes de prueba (testnets) y la interacción con *smart contracts* ya desplegados. Una de las características más útiles de este IDE es su análisis estático del código, pudiendo así identificar posibles errores de programación o vulnerabilidades de seguridad. Un aspecto crucial para el desarrollo de *smart contracts* donde los errores pueden suponer consecuencias financieras significativas.

Además, Remix se integra con herramientas y *plugins* adicionales, ofreciendo un entorno más rico y extenso permitiendo conectarse con herramientas y servicios como MetaMask, Truffle y Ganache los cuales han sido de gran utilidad en mi proyecto. Remix ha tenido una gran protagonismo en el desarrollo de mi proyecto permitiéndome iterar rápidamente a través de diferentes versiones de contratos inteligentes. Pudiendo ejecutar pruebas unitarias con diversas redes Ethereum como Rinkeby y Goerli, indispensable para validar la lógica del *smart contracts* antes de su despliegue final. Dándome una gran capacidad de testeo en un entorno controlado pero realista el cual ha sido crucial para la corrección de errores y la optimización del uso de gas, asegurando así la eficiencia y seguridad de los contratos inteligentes desarrollados.

Infura

Infura es una plataforma que proporciona una API escalable y de alta disponibilidad para acceder a la *blockchain* de Ethereum, entre muchas otras. Esta plataforma permite desplegar contratos inteligentes en la *blockchain* sin necesidad de mantener un nodo propio lo que representa un ahorro significativo en timpo y recursos. Para ello, Infura proporciona ´endPoints' de API que facilitan la lecutra y escritura de datos en la *blockchain*. Infura es una propuesta de gran valor para el desarrollo de dApps como la presente, ya que facilita el despliegue y la interacción de contratos inteligentes, asegurando

una alta disponibilidad. Esto garantiza que las aplicaciones puedan realizar y recibir transacciones en cualquier momento. Del mismo modo, a medida que el proyecto crece, Infura se adapta a las necesidades de escalabilidad, permitiendo que la aplicación pueda manejar un mayor número de solicitudes de manera eficiente.

Truffle

Tras una fase inicial de desarrollo de los *smart contracts* usando Remix, la transición al uso de Truffle ha marcado un punto de inflexión en la complejidad de mi proyecto. Truffle consiste en una suite de desarrollo avanzada para Ethereum, ofreciendo un conjunto de herramientas diseñadas para facilitar el desarrollo, gestión y despliegue de *smart contracts*. Truffle ofrece un entorno de desarrollo estructurado generando una jerarquía de carpetas para favorecer la implementación de proyectos *blockchain* complejos. Ahorra mucho tiempo con su sistema de migraciones y scripts de despliegue el cual automatiza y simplifica el proceso de lanzamiento de contratos.

Aunque Remix es una excelente herramienta, Truffle eleva la posibilidad de hacer pruebas unitarias a otro nivel con un marco de prueba mucho más sofisticado, permitiendo ejecutar test en Solidity o JavaScript.Esta capacidad ampliada para realizar pruebas unitarias contribuye de manera crucial a la calidad y seguridad del código de los *smart contracts*.

Finalmente, uno de las mayores ventajas de usar Truffle en el desarrollo de aplicaciones descentralizadas es en cómo simplifica el proceso de unir el trabajo que se realiza en el backend con el frontend. Truffle proporciona herramientas que que promueven una conexión mas directa y menos propensa a errores entre *smart contracts* y las aplicaciones móviles.

Ganache

Ganache funciona como un nodo de Ethereum personal, permitiendo a los desarrolladores simular un entorno de *blockchain* que opera localmente ofreciendo un espacio seguro y controlado para experimentar sin ningún costo real ni tiempos de espera al contrario que pasa con las redes públicas de Ethereum. Por defecto, proporciona diez cuentas cargadas con 1000 ETH cada una junto con sus correspondientes claves públicas y privadas, aunque esta configuración puede ajustarse según las necesidades del proyecto.

Disponible tanto como una herramienta en línea de comandos o como una aplicación de escritorio. Ganache proporciona una vista de las transacciones,

bloques y estado de la red, mostrando el feedback en tiempo real, vital para realizar una iteración rápida.

La integración de Ganache con Truffle facilita aún más su utilidad, estableciéndose como el entorno de desarrollo local predeterminado dentro del ecosistema Truffle. De esta forma se permite testear la eficiencia de los contratos ajustando parámetros como el tiempo de bloque para simular diferentes condiciones de red.

MetaMask

MetaMask es una extensión de navegador y una aplicación móvil que permite a los usuarios interactuar con la *blockchain* de manera segura y sencilla. Actúa como puente entre los navegadores web y la *blockchain*. Su funcionamiento es análogo a una cartera digital, permitiendo a los usuarios almacenar sus cuentas de Ethereum así como el envío de criptomonedas.

Al conectarse a dApps, los usuarios pueden usar sus cuentas de MetaMask para autenticarse eliminando la necesidad de ingresar claves privadas manualmente. Por otro lado, aplica una capa adicional de seguridad al encriptar la información del usuario y almacenar las claves privadas directamente en el dispositivo del usuario. Esto asegura que solo el usuario tenga acceso a sus fondos y datos.

La integración de MetaMask al proyecto no solo mejora la experiencia del usuario final, proporcionando una forma más segura e intuitiva de acceder a sus activos, sino que también agiliza el desarrollo de mi proyecto al simplificar la manera en que los usuarios se conectan a la dApp.

Finalmente una característica definitiva de Metamask a diferencia de otras billeteras, es su capacidad para agregar redes personalizadas, de gran utilidad para el testeo de la aplicación, permitiendo la conexión a entornos específicos como el entorno local proporcionado por Ganache o la tesnet de Sepolia.

WalletConnect

WalletConnect es un protocolo de código abierto que facilita la conexión segura entre billeteras móviles y aplicaciones descentralizadas. Cuando un usuario quiere interactuar con una dApp, WalletConnect genera un código QR o un enlace directo utilizado por la billetera descargada en el dispositivo del usuario. Esto establece una conexión cifrada y autoriza a la dApp para enviar solicitudes de transacción a la cartera.

El usuario puede entonces administrar las transacciones directamente desde su billetera, garantizando que mantiene el control total sobre sus claves privadas y, por ende, sobre sus activos. De esta manera se elimina la necesidad de ingresar claves privadas en la dApp.

WalletConnect es compatible con una amplia variedad de billeteras, lo que la convierte en una herramienta muy poderosa, mejorando la accesibilidad y experiencia del usuario.

Node.js

Node.js es un entorno de ejecución en JavaScript, que tradicionalmente ha sido un lenguaje de programación del lado del cliente, para desarrollar aplicaciones del lado del servidor. Es conocido por su capacidad para manejar operaciones asíncronas y por su escalabilidad siendo de gran popularidad en el desarrollo de aplicaciones web modernas. Una de las ventajas de de Node.js es su ecosistema de paquetes gestionados por NPM (Node Package Manager), que proporciona acceso a miles de librerías.

En el proyecto se ha usado de manera frecuente, más allá de ser un requisito para utilizar Truffle, a través del comando ïpm installp̈ara descargar librerías y paquetes necesarios tanto para el desarrollo del frontend como el del backend. Bien es así, que aunque React Native sea el *framework* principal del desarrollo del frontend, la gestión de sus dependencias, librerías adicionales se realiza mediante npm, el cual opera sobre Node.js. Por ejemplo el uso de la biblioteca 'Ethers', una de las mas importantes del proyecto ya que es fundamental para interactuar con la *blockchain*, se ha instalado y gestionado usando npm. Por tanto, Node.js ha sido una pieza crucial en el desarrollo del proyecto, demostrando su valor no solo como una plataforma de desarrollo del lado del servidor, sino también como un eje central para la administración de paquetes y librerías en el desarrollo de aplicaciones completas.

EtherScan

EtherScan ofrece una plataforma web que permite explorar los bloques de la red de Ethereum. Esta herramienta permite seguir la actualización de la red Ethereum en tiempo real, observando cómo se añaden nuevos bloques y el conjunto de transacciones de cada uno. También permite buscar específicamente una transacción que se ha cometido a lo largo del tiempo mediante su función hash. A parte de mostrar las transacciones en tiempo real, EtherScan permite desglosar cada transacción para ofrecer detalles como

el número de bloque, la hora exacta de la confirmación, y datos específicos sobre las acciones dentro de las transacción, tales como transferencias de *tokens* ERC-721, el valor transferido, las tarifas de gas, etcétera. EtherScan también sirve como un centro de análisis, en su plataforma podemos encontrar información relevando sobre el estado de la red, cómo estadísticas en tiempo real y datos históricos sobre la dificultad del minado, el precio del gas y otras métricas que pueden ser da gran utilidad para entender el comportamiento de la app en un momento determinado. EtherScan ha tenido un gran protagonismo en la etapa final de mi aplicación donde se ha desplegado el contrato inteligente en una red real de prueba. Con esta herramienta se ha monitoreado la ejecución y rendimiento del contrato, asegurando su correcto funcionamiento y fiabilidad. Por otro lado, ha permitido experimentar y observar la eficiencia del contrato dependiendo de las fluctuaciones en el precio del gas y otras condiciones de la red.

Sepolia PoW Faucet

En las primeras implementaciones del proyecto se ha utilizado ETH ficticios que se generaban en un contexto local, pero en las versiones finales del proyecto y en su despliegue en una red de prueba real ha sido necesario obtener ETH de prueba reales para poder interaccionar con la *blockchain*. Al ser ETH de prueba, estos carecen de valor y por lo tanto no se pueden comprar como se haría con ETH de la *mainnet*, por lo tanto las únicas formas de conseguirlos es minando o que otro usuario te los proporcione. Convertirse en un minero para obtener ETH de prueba es una opción más compleja, por lo que el uso de las faucet reduce en gran cantidad esta barrera de entrada.

Existen diversas faucets que proveen una cantidad de ETH de prueba diaria, pero en la mayoría para verificar la autenticidad del usuario se requiere que dicho usuario cuente con una pequeña cantidad de ETH reales en su billetera y por lo tanto obligan a hacer una pequeña inversión. Sin embargo, existe una faucet llamada Sepolia PoW Faucet que verifica la autenticidad del usuario mediante el mecanismo de prueba de trabajo (PoW), eliminando la necesidad de poseer ETH reales. Este enfoque requiere que los usuarios realicen un trabajo de cómputo, específicamente generando hashes que cumplan con ciertos criterios preestablecidos para demostrar su esfuerzo y recibir ETH de prueba a cambio.

4.2. Bibliotecas

Se muestra a continuación las bibliotecas usadas a lo largo del desarrollo del proyecto.

- Bibliotecas para implementación *blockchain*:
  - **OpenZeppelin:** OpenZeppelin es una biblioteca para el desarrollo seguro de *smart contracts* en Ethereum. Ofrece implementaciones auditadas y probadas para minimizar riesgos de seguridad. Su uso ha sido crucial para el desarrollo de mi proyecto, usando el estándar ERC721 para la representación de los contratos como tokens no fungibles (NFTs) asegurando que la aplicación cumpla con los estándares de seguridad.
  - **ethers:** Biblioteca caracterizada por su simpleza, ligereza y seguridad para interactuar con la red de Ethereum. Proporciona funcionalidades para crear clientes que requieren comunicarse con la *blockchain* de Ethereum, permitiendo la gestión de billeteras, la construcción y firma de transacciones, la conexión a diferentes nodos de Ethereum a través de varios proveedores como JSON RPC o Infura, y la interacción con contratos inteligentes.
  - **web3:** Biblioteca formada por un conjunto de módulos que permite a las aplicaciones cliente interactuar con una *blockchain* local o remota a través del protocolo Ethereum. Cuenta con un conjunto de características más amplio que la biblioteca ethers, haciéndola algo más compleja, pero en la practica son bibliotecas muy parecidas.
  - **react-native-get-random-values:** Polyfill que proporciona valores aleatorios seguros para criptografía en React Native, crucial para generar claves y tokens únicos de forma segura.
  - **ethersproject/shims:** Facilita la compatibilidad de ethers.js en React Native, permitiendo el desarrollo de dApps mediante la provisión de funciones de criptografía y *blockchain* en dispositivos móviles.
  - **text-encoding:** Es un polifill que implementa la API 'TextEncoder' y 'TextDecoder'. Estas API permiten la codificación y decodificación eficiente de texto en formatos como UTF-8.
- Bibliotecas para implementación billetera:

- **metamask/sdk:** Biblioteca que proporciona una conexión segura y fluida desde la dapp a la extensión del navegador o app móvil de MetaMask.
- **WalletConnect:** protocolo estándar que facilita la conexión segura entre billeteras móviles y aplicaciones descentralizadas (dApps) mediante el escaneo de un código QR o un enlace directo.
- **walletconnect/react-native-compat:** Asegura la compatibilidad de WalletConnect con React Native, simplificando la integración de WalletConnect en aplicaciones móviles
- **Viem:**Permite a los desarrolladores interactuar con la *blockchain* de Ethereum, incluyendo abstracciones de la API JSON-RPC, interacción con contratos inteligentes, implementaciones de billeteras y firmas, utilidades de codificación/descodificación, y más. Viem actúa como una base sólida sobre la cual se construyen herramientas más complejas, como Wagmi.
- **Wagmi:** Wagmi Core es un envoltorio sobre Viem que proporciona funcionalidad multi-cadena a través de la configuración de Wagmi y manejo automático de cuentas mediante Conectores. Resuelve problemas comunes como la conexión de billeteras, soporte multi-cadena, envío de transacciones, escucha de eventos y cambios de estado, y refresco de datos de *blockchain*.
- **web3modal/wagmi-react-native:** adapta WAGMI y Web3Modal para React Native.

- Bibliotecas para implementación base de datos:

  - **Firebase:** Biblioteca que proporciona acceso a los diversos servicios de Firebase. Es el punto de partida para la integración de cualquier funcionalidad de Firebase.
  - **firebase/firestore:** Biblioteca que permite la integración con FireStore. Facilita las operaciones de consultas, actualizaciones y escucha de cambios en la base de datos.
  - **react-native-firebase/app:** Módulo principal de Firebase para React Native, necesario para inicializar y configurar Firebase en la aplicación móvil.
  - **react-native-async-storage:** Biblioteca para almacenamiento asincrónico y persistente de datos, ideal para almacenar preferencias de usuario y configuraciones.

- Bibliotecas para la navegación de la aplicación:

  - **@react-navigation/native:** Biblioteca que vincula la funcionalidad de navegación de React Navigation con las APIs nativas del entorno de React Native. Proporcionando la funcionalidad para el enrutamiento y navegación en aplicaciones móviles.
  - **react-navigation/native-stack:** Ofrece una experiencia de navegación fluida y optimizada mediante el uso de APIs nativas para la navegación entre pantallas en aplicaciones React Native.
  - **react-native-tab-view:** Biblioteca que ofrece una solución para pestañas que se pueden deslizar entre sí, soportando la navegación basada en gestos.
  - **react-navigation/bottom-tabs:** Biblioteca para crear barras de navegación en la parte inferior de aplicaciones móviles, facilitando la navegación entre vistas.
  - **react-native-gesture-handler:** Biblioteca utilizada para el manejo avanzado de gestos. Proporcionando soporte para una variedad de gestos, como toques, arrastres y deslizamientos entre otros.

- Bibliotecas para el resto de funcionalidades:

  - **axios:** Biblioteca cliente HTTP basada en promesas para el navegador y para Node.js. Permite hacer solicitudes HTTP a servidores externos de manera sencilla y eficaz.
  - **react-native-qrcode-svg:** Biblioteca de React Native que permite la generación de códigos QR en formato SVG.
  - **expo-barcode-scanner:** Módulo de Expo que ofrece funcionalidades de escaneo de códigos de barras y QR en aplicaciones React Native.
  - **expo-local-authentication:** Módulo de Expo que proporciona métodos para permitir la autenticación biométrica como huella digital o reconocimiento facial.
  - **react-native-community/datetimepicker:** Componente de React Native que proporciona interfaces de usuario para seleccionar fechas y horas. Facilita la interacción del usuario con la aplicación.

- **react-native-picker/picker:** Biblioteca que ofrece un componente de desplegable que permite a los usuarios seleccionar un item.
- **country-state-city:** Biblioteca que proporciona listas de países, estados y ciudades. Utilizado para mostrar en formularios donde se tiene que elegir una opción.
- **react-native-chart-kit:** Biblioteca que permite la integración de gráficos y visualizaciones en aplicaciones React Native

4.3. Otras herramientas

Esta sección incluye herramientas y programas que tienen una relevancia secundaria en el proyecto.

- **icons.expo.fyi:** Recurso en línea ofrece una visualización completa de los íconos disponibles en la biblioteca @expo/vector-icons. Facilita la integración rápida de los ícono al proporcionar el nombre exacto y el código de importación necesario para cada ícono.
- **CoinGecko:** Página web para el análisis de criptomonedas. Se utiliza su API en el proyecto para acceder a la información actualizada sobre el precio del Ethereum.
- **TEXStudio:** IDE para LaTeX.
- **Visual Studio Code:** Editor y depurador de código empleado junto con alguna extensión.
- **GitLab:** Plataforma de control de versiones,
- **Git Bash:** Emulador de línea de comandos para Windows que proporciona herramientas de Git.
- **Notion:** Organizador de tareas utilizado para tomar notas.
- **Mendeley:** Gestor de referencias para organizar investigación.

5. Aspectos relevantes del desarrollo del proyecto

En este apartado se pretende resumir los aspectos más relevantes que se han tratado y cómo se han resuelto las diferentes dificultades encontradas a lo largo de su desarrollo.

5.1. Dificultades encontradas en el desarrollo

Este proyecto se ha caracterizado por las diversas dificultades que se han encontrado primordialmente exacerbadas por las incompatibilidades de diversas librerías y herramientas en el desarrollo móvil. A continuación, describo alguno de los problemas más significativos y cómo los he abordado.

Incompatibilidad con biblioteca Web3

En el momento en el que se intentó realizar la primera conexión con el contrato desde la aplicación móvil, surgió un problema con la biblioteca Web3, esencial para la interacción entre el frontend y el contrato inteligente. El error se producía a la hora de ejecutar la aplicación de forma nativa en el dispositivo móvil. El error describía un problema 'ReferenceError' indicando que la propiedad 'EventTarget' no existía en el motor de JavaScript Hermes utilizado por React Native.

```
ReferenceError: Property 'EventTarget' doesn't exist, js engine:
hermes ERROR Invariant Violation: 'main' has not been registered.
```

Sin embargo, desplegando la aplicación en un entorno web desde el navegador de mi ordenador esta biblioteca no producía ningún error y se consiguió establecer conexión exitosamente con el contrato inteligente. Este fue un problema que se alargó durante los dos siguientes *sprints* y se investigó de forma reiterada. Temiendo que se debía a un error de incompatibilidad con las versiones de las bibliotecas en mi proyecto, se fue probando diferentes combinaciones hasta que usando la versión 4.1 de la biblioteca Web3 ese error desapareció. El problema no se había solucionado ya que dicho error solo desapareció dejando paso a un nuevo error, en este caso el error describía un problema de reconocimiento similar al anterior pero con la propiedad 'TextEncoder' la cual es parte de la API de codificación que proporciona una forma de codificar texto en varios formatos, comúnmente utilizados en entornos web.

`ReferenceError: Property 'TextEncoder' doesn't exist, js engine: hermes.`

React native no implementa todas las API web estándar que se encuentran en los navegadores modernos, por lo que es necesario agregar dicha característica de forma manual. Para ello he hecho uso de un *polyfill*, que es una biblioteca que pretente proporcionar una implmentación alternativa a la API de funciones nativas. Por tanto, se ha instalado 'text-encoding', que es un polyfill para la API de 'TextEncoder'. Seguidamente se ha creado un nuevo archivo en la raíz del proyecto al que he llamado 'globals.js', en este archivo se ha asignado 'TextEncoder' de 'text-encoding' a 'global.TextEncoder'. Es decir, que he agregado 'TextEnoder' al objeto global 'global' que actúa de forma similar al objeto 'window' en los navegadores.

```
global.TextEncoder = require('text-encoding').TextEncoder;
```

Luego importando 'globals.js' en mi archivo 'App.js' aseguro que 'TextEncoder' este disponible globalmente antes que cualquier otro objeto que dependa de él, evitando errores de referencia.

Tamaño de contrato muy grande

Durante el desarrollo en Solidity del contrato inteligente surgió un desafío inesperado relacionado con el tamaño del código del contrato. El proceso de desarrollo del contrato inteligente se ha realizado de forma iterativa, empezando con funciones básicas y gradualmente se fue implementando funcionalidades más complejas. Después de cada adición significativa, se procedía a migrar y testear el contrato para asegurar su correcto funcionamiento y rendimiento.

En una de estas iteraciones, al intentar migrar el contrato, apareció un error que exponía un problema de que el código había superado los 24576 bytes de tamaño, un límite establecido por la actualización "Spurious Dragon"de Ethereum. Este límite fue introducido para mitigar riesgos de ataques de denegación de servicio (DoS) que podrían ocurrir si se permitiera desplegar contratos de tamaño excesivamente grande.

code size is 26466 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low runs"value!), turning off revert strings, or using libraries.

Con el objetivo de que el contrato sea menor de esos 24576 bytes y así mantener toda la funcionalidad en un único contrato facilitando así su gestión y uso, se aplicaron diversas técnicas. En una primera instancia, se hizo una revisión del código para identificar y eliminar redundancias, optimizar las funciones existentes, y donde fue posible, integrar librerías externas que reemplazaran bloques de código repetitivos. Por otro lado, se habilitó el optimizador del compilador de Solidity, ajustando el número de *runs* a un valor bajo, con la intención de reducir significativamente el tamaño del bytecode generado al optimizar la compilación con base en la frecuencia esperada de ejecución del código.

Sin embargo, a medida que el proyecto evolucionaba y con la aparición de nuevas funcionalidades para implementar, esta estrategia de optimización dejó de ser sostenible y fue necesario modularizar el código. Para ello se dividió el contrato en múltiples contratos más pequeños que interactúen entre sí mediante herencia o mediante llamadas a funciones de otros contratos. La modularización no solo ayudó a gestionar mejor el tamaño del contrato, sino que también mejoró la mantenibilidad y la actualización del código.

Problemas implementación MetaMask

La integración de una billetera de terceros en el proyecto ha presentado un gran desafío, con la idea de implementar una solución que permitiera al usuario gestionar sus cuentas, se exploraron diferentes posibilidades. En este caso, se optó por la opción implementar MetaMask, debido a ser la billetera ampliamente más conocida y usada por la comunidad. Por lo tanto, la idea era usar los servicios de MetaMask para que los usuarios pudieran acceder a la aplicación móvil con su cuenta personal y más adelante poder interactuar con dicha cuenta.

Inicialmente, se hicieron diferentes pruebas para establecer una conexión entre MetaMask y mi aplicación. Estas pruebas se centraron en utilizar la extensión de navegador de MetaMask con el objetivo de establecer la conexión, donde se consiguió establecer una conexión estable y continua permitiendo interactuar con la billetera libremente. Aunque esta implementación solo aseguraba el funcionamiento de la aplicación en un entorno de navegador.

Partiendo de su correcto funcionamiento en navegador se quiso expandir la funcionalidad para que fuera compatible en la aplicación móvil. Este paso resultó ser considerablemente más complejo debido a la incompatibilidad directa de las extensiones de navegador en los entornos móviles. Inicialmente se intentó usar el SDK de MetaMask, recientemente lanzado, empleando un enfoque de enlaces profundos para interacción directa con la aplicación de MetaMask en dispositivos móviles. La idea era, que una vez el usuario hubiera configurado correctamente sus cuentas en la app de Metamask, cuando quisiera realizar alguna operación que requeriese de la billetera, la aplicación móvil generaría un URI de MetaMask que abriría directamente la aplicación de MetaMask instalada en el dispositivo del usuario. Desde allí el usuario podría autorizar la operación. No obstante, este método enfrentó grandes obstáculos técnicos, impidiendo establecer una conexión con la aplicación móvil de MetaMask.

A raíz de este problema se investigaron alternativas para poder ofrecer una conexión estable entre la aplicación y la billetera móvil. De esta forma se planteó la utilización de 'WalletConnect', un protocolo de código abierto altamente utilizado por la comunidad y que facilita la comunicación entre billeteras móviles y aplicaciones descentralizadas mediante enlaces seguros. Tras un largo proceso de desarrollo, se logró implementar WalletConnect con la posibilidad de establecer conexiones no solo con MetaMask sino con otras cientos de billeteras.

Sin embargo, la aplicación aún no era capaz de realizar operaciones a través de esta nueva configuración. Durante un par de *sprints* dedicados principalmente a este problema, se identificó la necesidad de una revisión completa de la arquitectura de la aplicación. Para ello se abandonó la biblioteca Web3, que había sido la pieza fundamental de conexión entre los contratos inteligentes y el frontEnd. En su lugar, se adaptó la aplicación para usar las bibliotecas Ethers, Wagmi y Viem, que ofrecían una mayor flexibilidad y compatibilidad con WalletConnect.

Esto proceso fue bastante tedioso ya que implicó establecer una nueva conexión con el contrato mediante el nuevo provedor, así como actualizar todas las llamadas al contrato inteligente para que estas puedan ser ejecu-

tadas correctamente por la billetera configurada en WalletConnect. Esta transformación no solo resolvió los problemas iniciales, sino que también se mejoró la accesibilidad de la aplicación permitiendo la integración de cientos de billeteras.

Incompatibilidad con Ganache y Metamask

Entre los problemas que han surgido debido a incompatibilidades con la aplicación móvil, se ha identificado un error el cual imposibilita crear y conectar una red Ganache desde la aplicación móvil de MetaMask. Al igual que sucedía con la biblioteca Web3, este problema no existe en el desarrollo web.

Como he mencionado anteriormente, para la ejecución de mi proyecto es necesario desplegar una red Ganache la cual normalmente se despliega en *localhost*. Sin embrago, para que el resto de dispositivos de la red puedan interactuar con ella, es neceario asignarle una dirección IP específica, en este caso la dirección IP de mi ordenador. Al desplegar Ganache se muestran datos importantes como la URL RPC y la ID de la cadena, las cuales son esenciales para configurar una red personaliada en MetaMask. A partir de esta configuración, Metamask permite importar cuentas disponibles en el entorno local de Ganache a través de su clave privada, replicando de esta forma un uso realista en el entorno de prueba de como se realizarían las interacciones mediante la billetera, la cual funciona como interfaz para interactuar con la blockchain.

Sin embargo, he encontrado un problema al realizar este proceso en la aplicación móvil de MetaMask, a diferencia de la extensión de nevagador, la aplicación móvil no soporta de manera uniforme las redes HTTP Y HTTPS, devolviendome el siguiente error:

```
Could not fetch chain ID. Is your RPC URL correct?
```

Este mensaje indica que no se puede recuperar el ID de la cadena, el cual es un identificador único de cada blockchain, crucial para la seguridad, utilizado por las billeteras para asegurar que interactúan con la red correcta. Este problema radica en que comúnmente, Ganache despliega la red usando HTTP en un entorno local, lo cual es apto para la extensión de navegador, pero no para la aplicación móvil ya que requiere HTTPS. Este es un problema recurrente para muchos desarrolladores y aún no cuenta con una solución directa. Una solución temporal es crear un túnel seguro y exponer la red Ganache local usando herramientas como ngrok, que proporciona una URL HTTPS temporal. De esta forma se expone un servidor local a internet a

través de un túnel seguro, permitiendo así que la aplicación de MetaMask establezca una conexión segura. Bien es cierto, que esto soluciona en cierta parte el problema, pero es solo una solución temporal y no es ideal para entornos de producción.

Desafíos del despliegue en Mainnet

Dada la imposibilidad de utilizar Ganache en conjunto con MetaMask,se hizo necesario a buscar una solución al problema para poder integrar una billetera en el proyecto. En el proceso de investigación para resolver este problema, se dedujo que la única opción para poder usar MetaMask era desplegar el contrato en una red blockchain real.

Para poder integrar MetaMask en el proyecto y realizar operaciones sin que suponga ningún coste económico real, se decidió usar la red de Sepolia, que es una red de prueba de Ethereum. Esta idea, finalmente acabó complicando el proyecto notablemente, siendo necesaria una adaptación de todo el código que llevó varías semanas de trabajo. Para usar la red de Sepolia, fue necesario desplegar el contrato en dicha red. El primer paso fue obtener ETH de Sepolia, necesarios para cubrir los costes del despliegue, lo cual es un proceso lento debido a las pequeña cantidad que se puede obtener diariamente. Además, para interactuar con una red real, fue esencial establecer un proveedor de servicios de terceros. Para esta tarea se seleccionó Infura. La cual mediante una URL permite interactuar con la red Ethereum sin la necesidad de ejecutar un nodo de Ethereum propio.

Seguidamente fue necesario llevar a cabo la configuración de Truffle para poder desplegar el contrato en esta red, haciendo uso de la URL proporcionado por Infura y de una cuenta Ethereum con suficientes fondos. Una vez desplegado el contrato, es necesario establecer conexión con el contrato desde la aplicación, proceso que también se hace mediante los servicios de Infura. Para este paso, debido a las grandes dificultades A pesar de los múltiples problemas enfrentados con la biblioteca Web3 durante todo el desarrollo, y debido a la incompatibilidad con WalletConnect, para este paso se decidió utilizar la biblioteca Ethers, que proporcionó una integración más fluida y estable.

Por otro lado, para el correcto funcionamiento de la aplicación en una red blockchain real, fue necesario modificar todo el código donde se realizase alguna operación sobre el contrato. Una vez configurado todo, y en combinación con WalletConnect, se realizaron las primeras operaciones. Sin embargo, los tiempos de espera para la confirmación de acciones y las operaciones

de lectura se vieron significativamente afectados por la latencia de la red principal, lo cual requirió de ajustes adicionales en la aplicación para mejorar la fluidez y la experiencia del usuario.

A pesar de los éxitos iniciales, surgieron problemas al desplegar el contrato inteligente nuevamente en la red Sepolia, la API de Infura reportó que se había superado el límite de llamadas. Este fue un problema inesperado, debido que hasta el momento se había logrado desplegar el contrato en varias ocasiones. No puedo asegurar el origen del problema, pero seguramente tenga que ver con que días antes la empresa reportó una alta demanda y se realizaron limitaciones en los servicios de IPFS. Hasta día de hoy la API sigue devolviendo dicho error, por lo que esto llevó a explorar alternativas como Alchemy, que ofrecía límites más generosos en sus consultas. A pesar de lograr desplegar el contrato usando Alchemy, se encontraron dificultades para interactuar desde la aplicación con el contrato a través de este proveedor.

Ante la acumulación de desafíos y la falta de tiempo, se tomó la decisión de regresar a la implementación inicial utilizando Ganache, lo que resultó en una aplicación más estable y manejable para la detección de errores, pero sin la posibilidad de implementar el inicio de sesión con una billetera. Este cambio, aunque marcó un retroceso en a etapas anteriores del diseño e implementación de la aplicación, permitió un contacto valioso con la red Ethereum real y abrió caminos para el futuro desarrollo del proyecto, dejando establecida una base de código preparada para operar en este entorno.

5.2. Incompatibilidad Firebase

Durante la fase inicial del proyecto se optó por implementar una base de datos para la autenticación de los usuarios. Se decidió usar Firebase, debido a que es una herramienta muy poderosa y robusta, y mi interés de aprender a utilizarla. Entre las funcionalidades previstas estaban la autenticación de usuarios, el almacenamiento de datos y un servicio sólido de mensajería para funciones como el restablecimiento de contraseña y la verificación de dos factores. Además, se contempló la posibilidad de permitir iniciar sesión con google, una funcinalidad nativa de esta base de datos.

En una primera instancia se implementó correctamente las funcionalidades de registro, verificación y recuperación de contraseña. Sin embargo, en el momento que se importaba la biblioteca de Web3, la aplicación compilaba sin errores, pero el proceso de autenticación de usuarios fallaba, retornando un error sin una solución clara tras varias investigaciones.

A medida que el proyecto evolucionaba, surgió la idea de establecer un inicio de sesión usando billeteras como MetaMask y más adelante WalletConnect. Considerando esta integración, mantener la autenticación de Firebase parecía redundante, por lo que se decidió prescindir de la idea de realizar la autenticación de usuario con firebase en favor de la autenticación basada en billeteras.

No obstante, en las últimas semanas de desarrollo del proyecto como bien se ha comentado anterioridad, tras enfrentar múltiples dificultades con la implementación de las billeteras y descartar dicha posibilidad, la necesidad de implementar una autenticación de usuario resurgió. Por lo tanto, se reanudó la investigación sobre la incompatibilidad de Firebase y la biblioteca Web3, debido a un error específico que se desencadenaba por la presencia de Web3 en el proyecto, afectando la verificación de usuarios.

Tras un análisis y varias pruebas, se ideó una solución que consistía en implementar la importación 'perezosa' de Web3 (ver imagen 5.2). Esto implica cargar Web3 sólo después de que la autenticación de usuario fuera exitosa. Para llevar a cabo esta estrategia, se utilizó la técnica de 'dynamic imports' de JavaScript, que permite cargar módulos de manera condicional y asíncrona. Para ello, en vez de importar la biblioteca Web3, se llamará a este fragmento de código, el cual primero verifica si ya existe una instancia de Web3, devolviendo esta instancia si está disponible. En caso de que no haya una instancia previa, se procede a importar Web3 de manera dinámica solo cuando el usuario necesite realizar una acción que involucre dicha biblioteca. Además, el uso de promesas asegura que la carga del módulo se maneje de manera asíncrona sin interrumpir la interacción del usuario.

Este ajuste no sólo resolvió el problema técnico, sino que también permitió mantener el diseño inicial del proyecto, aprovechando la escalabilidad, seguridad y facilidad de uso de Firebase para la gestión de usuarios, mientras se mantiene la capacidad de integrar Web3 para futuras funcionalidades relacionadas con blockchain.

Code - EtherProvider.js

```
1  let web3Instance = null;
2
3  export async function getWeb3() {
4      if (!web3Instance) {
5          const { default: Web3 } = await import("web3");
6          const Url = "http://192.168.1.33:8545";
7          web3Instance = new Web3(new Web3.providers.HttpProvider(Url));
8      }
9      return web3Instance;
10 }
```

Figura 5.1: Importación perezosa Web3.

6. Trabajos relacionados

6.1. OpenLaw

OpenLaw es una plataforma que combina contratos inteligentes con protocolos legales convencionales. Desarrollada para mejorar la eficiencia, seguridad y transparencia en la gestión de acuerdos legales, OpenLaw se posiciona en la vanguardia de la intersección entre la ley y la tecnología blockchain.

La plataforma usa la red Ethereum para registrar cada contrato en la *blockchain* garantizando que una vez el contrato es firmado, este permanezca inalterable minimizando las posibles disputas sobre el y permitiendo una ejecución automática de los términos acordados.

OpenLaw integra tecnologías de firma digital, facilitando la ratificación de acuerdos sin necesidad de encuentros físicos. En regiones donde los contratos inteligentes y las firmas digitales están legalmente reconocidos, OpenLaw ofrece una herramienta poderosa y conforme a la ley. Sin embargo, en jurisdicciones sin una regulación clara sobre el uso de la tecnología *blockchain* en el ámbito legal, puede existir incertidumbre sobre la validez y ejecución de estos acuerdos.

En la imagen 6.1 se puede observar como se ratificaría un contrato usando OpenLaw.

6.2. Proyectos

Aplicación web para operar con contratos digitales a través de blockchain

Figura 6.1: Ejemplo de contrato vinculante usando OpenLaw.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.