# Universidad San Jorge

# Escuela de Arquitectura y Tecnología

# Grado en Ingeniería Informática

## Proyecto Final

## Notario Digital en Blockchain

**Autor del proyecto: Álvaro Tomás Lozano**

**Directora del proyecto: Mª Francisca Pérez**

**Zaragoza, 09 de septiembre de 2021**

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma

Fecha

09/09/2021

## Dedicatoria y Agradecimiento

Primero, quiero agradecerles a mi padre y a mi madre la oportunidad que me han brindado para estudiar lo que he querido durante 5 años y apoyarme en todo momento de mi vida. Gracias por vuestro esfuerzo y cariño que me habéis dado estos años. También, a mis familiares cercanos que han estado ahí y se han portado como una verdadera familia en tiempos difíciles.

Agradecer a todos mis amigos y compañeros que he hecho durante el transcurso de la carrera, que me han aportado tantos momentos buenos y me han apoyado en los malos. En especial a Javier Jiménez por ser un buen amigo y darme su ayuda cuando lo necesitaba.

Agradecer a los profesores de la Universidad San Jorge que he tenido durante el transcurso de mi carrera por el conocimiento y la ayuda que me han aportado, que siempre me han apoyado cuando lo que he necesitado y han tenido paciencia conmigo hasta en los momentos más duros.

Agradecer a África Domingo y a Mª Francisca Pérez por ser ambas unas excelentes tutoras.

Agradecer a Miguel Ángel Barea y Rodrigo Casamayor su apoyo y ayuda durante el desarrollo de todo este proyecto, y por aportar su confianza en mis capacidades para lograrlo.

Gracias a todos.

## Contenido

## Resumen

La notaría digital aporta una solución tecnológica respecto a cómo se puede firmar un documento o cerrar un acuerdo de suma importancia que requiera la presencia de un notario y las partes firmantes. Gracias a los avances tecnológicos, se ha creado la firma digital, siendo esta una firma única e irremplazable. Con la tecnología del blockchain, ahora somos capaces de almacenar ese documento firmado digitalmente en una red inmutable y descentralizada, dando lugar a un sistema de gran seguridad y fiabilidad.

Los objetivos de este trabajo han sido analizar las diferentes tecnologías de blockchain existentes y su integración con los sistemas actuales de identidad digital, definir y desarrollar la arquitectura necesaria para el uso del sistema, implementar los *APIs* e interfaces básicos para el manejo e integración del sistema, y analizar las posibles oportunidades y modelos de negocios derivados. El trabajo ha logrado los objetivos planteados comenzando con el estudio de las tecnologías de blockchain y ha acabado con la implementación de una web con todas las funcionalidades básicas para almacenar y visualizar documentos en una red de blockchain.

## Palabras Clave

Blockchain, Ethereum, *Smart Contract*, criptomonedas, transacción, contrato.

## Abstract

The digital notary provides a technological solution regarding how you can sign a document or close an extremely important agreement that requires the presence of a notary and the signing parties. Thanks to technological advances, the digital signature has been created, this being a unique and irreplaceable signature. With blockchain technology, we are now able to store that digitally signed document in an immutable and decentralized network, resulting in a highly secure and reliable system.

The objectives of this work have been to analyze the different existing blockchain technologies and their integration with current digital identity systems, define and develop the necessary architecture for the use of the system, implement the APIs and basic interfaces for the management and integration of the system, and analyze the possible opportunities and derivative business models. The work has achieved the proposed objectives starting with the study of blockchain technologies and has ended with the implementation of a website with all the basic functionalities to store and view documents in a blockchain network.

# 1. Introducción

Un blockchain es una tecnología que imita a una cadena de bloques, los cuales son capaces de almacenar información y están conectados unos a otros. Cada bloque posee la dirección donde se encuentra el bloque que le precede y el que le sigue, a parte de los datos que se hayan podido almacenar en su interior, de modo que siempre estamos seguros de que los bloques están conectados. Estos bloques se distribuyen de manera descentralizada en una red de dispositivos, de forma que cualquiera puede disponer de la información, pero para modificarla se requiere una gran carga computacional por lo que se vuelve imposible para un usuario o un grupo de usuarios modificar su contenido y engañar al sistema.

Lo que hoy conocemos como blockchain tiene su origen años atrás. Al final la utilidad de una cadena de bloques es que esa información tenga una identidad única, dada por un sellado de tiempo que indique en qué momento fueron suministrados dichos datos y una certificación de que estos datos no han sido ni modificados ni alterados durante el transcurso del tiempo. Gracias a los avances tecnológicos en los últimos años se logró concebir un concepto de sellado de tiempo confiable y experimentar con ello en computadores.

En 1991, un grupo de investigadores crearon lo que por primera vez se conocería como el sellado de tiempo confiable, aplicando criptografía y firmas digitales, y haciendo que los certificados fuesen unidos, sabiendo qué certificado había sido emitido antes y cual después, siendo grabado en el documento. Esto sería olvidado hasta años después.

En 2009, el creador de Bitcoin, Satoshi Nakamoto, readaptaría el concepto del sellado de tiempo confiable para crear el primer blockchain y la primera criptomoneda del mundo. Gracias a esta nueva tecnología, comenzaron a aparecer nuevas criptomonedas que utilizarían sus propias redes de blockchain, añadiendo nuevas funcionalidades y características. Entre ellas, destacaría Ethereum, siendo la segunda criptomoneda más potente del mercado y con una tecnología que superaría al resto.

Con Ethereum llegarían los *Smart Contracts*, o contratos inteligentes, los cuales permiten interactuar con la red de bloques de Ethereum y aplicar una lógica detrás, haciendo de la cadena una herramienta de desarrollo muy potente que da lugar a un sinfín de posibles aplicaciones tecnológicas.

El objetivo de este trabajo ha sido desde un principio conseguir un producto mínimo viable en un estado base que permita realizar algo similar a una notaría digital utilizando las bases del blockchain, requiriendo estudiar las tecnologías que lo engloban para poder elegir adecuadamente cual es la más apta y conseguir así una aplicación que permita cargar documentos en la red y que sean consultados por los usuarios que lo requieran gracias a los contratos inteligentes.

Debido a las características del proyecto, se aplicó una metodología ágil basada en *sprints*, utilizando las reuniones de tutoría como hitos en el trabajo, analizando los problemas que hubiese con los resultados de las anteriores reuniones y estableciendo nuevas metas para la siguiente.

El resultado final concluye con una página web que implementa todo lo necesario para poder realizar los objetivos establecidos, permitiendo hacer de esta un producto personalizable para el cliente. Los usuarios podrán subir documentos a la red de blockchain de manera segura a través de una sencilla interfaz web que permite ver a cada usuario lo que ha subido, cargar nuevos documentos y generar una cartera nueva a su gusto.

# 2. Estado del Arte

## 2.1. Blockchain

En este apartado se explicará qué es el blockchain, cómo funciona, qué tipo de redes que hay, sus mecanismos, y se dará una pequeña introducción a los contratos inteligentes para luego poder entender el apartado técnico de los mismos. A parte, se analizarán algunas de las aplicaciones de notaría digital existentes y se justificarán las diferencias respecto a este proyecto.

### *2.1.1. ¿Qué es?*

El blockchain se define como "un libro de contabilidad digital que se distribuye entre varias ubicaciones para garantizar la seguridad y facilidad de acceso a nivel mundial, permitiendo a consumidores y proveedores conectarse directamente, eliminando la necesidad de un tercero". [1]

La verdadera función de la blockchain es generar un listado de transacciones histórico, similar a una base de datos, pero haciendo las transacciones inmutables una vez se ha creado una nueva.

Esta se distribuye a través de dispositivos, ya sean móviles, ordenadores personales o servidores, de modo que, gracias a una conexión a internet, la red puede distribuirse de forma equitativa entre todos los dispositivos que forman parte de ella. Todos distribuyen la blockchain a todos.

A su vez, que todos los dispositivos dispongan de la información de la red crea una amenaza a su contenido, dado que alguien podría modificar lo que hay en la red y todos recibirían el cambio. Para solucionar esto, la red de blockchain utiliza la criptografía y los algoritmos de prueba de trabajo o participación, los cuales se describirán en la sección 2.2.

Por su definición, mientras que en una red normal se dispone de un servidor que actúa de centro y al que los clientes le piden la información, la blockchain se distribuye como una red *peer-to-peer* o red entre par, similar al funcionamiento de BitTorrent, un protocolo diseñado para intercambiar archivos entre dispositivos, de modo que los usuarios puedan descargar archivos desde los dispositivos de otros usuarios sin necesidad de conectarse a un servidor

central, reduciendo así la carga que supone que múltiples usuarios descarguen el mismo archivo a la vez en un servidor.

Cualquier usuario que se una a la red a través de un dispositivo tiene que descargar una copia de la blockchain que se actualiza y sincroniza conforme el resto de los usuarios van interactuando con ella. Esta red es capaz de funcionar sin un ente o servidor que la aloje y aquellos usuarios que se unen pasan a ser nodos capaces de redistribuir la blockchain y validarla.

La gran ventaja de su descentralización es que no hay ningún ente gubernamental que pueda apropiarse de ella, por lo que está completamente despolitizada y ningún gobierno o país puede retirar la información que se ha grabado en ella. Esto supone un cambio también para los bancos, dado que con la creación de las criptomonedas y su valor estimado se pueden hacer intercambios monetarios entre usuarios de distintos países sin necesidad de pasar por una entidad bancaria, relegando el papel del banco a otro plano.

Aunque en principio la red está completamente descentralizada, su lógica se comparte entre los nodos, dado que entre ellos se acuerda un estado común de la blockchain y al final todos acaban teniendo el mismo estado de la red.

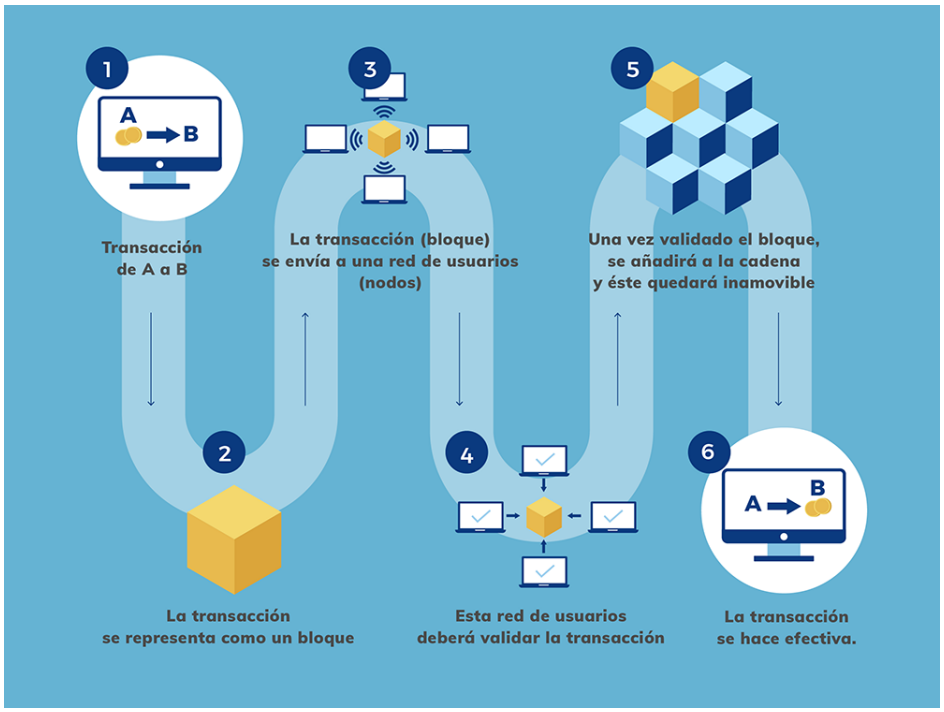


*Figura 1 – Proceso de validación de una transacción en blockchain. Fuente: https://www.gnarum.com/blockchain-hacia-nuevo-modelo-energetico/*

Debido a la inmutabilidad de la red, una vez se crea una transacción, esta no se puede deshacer, al menos de forma normal y para cualquier usuario estándar. Esta será validada siguiendo un proceso como el de la figura 1 y una vez sea efectiva se registrará permanentemente en la red. Aun con esto, la inmutabilidad no es del todo completa, ya que para que algo cambie respecto a cómo era originalmente, este cambio tiene que ser validado por los distintos usuarios de la red, por lo que para denegar un cambio aceptado en la blockchain tendrían que ponerse de acuerdo todos los usuarios de la red.

La integridad de la red les corresponde a los usuarios y no a un ente central, por lo tanto, cualquier usuario que intente modificar un bloque o una transacción requerirá de una validación de una gran parte de los usuarios para que esa modificación sea exitosa. Así se consigue que aquellos que intenten realizar actividades fraudulentas en la red sean completamente rechazados y desalentados a intentar un nuevo ataque a la red. Es por ello por lo que la tecnología de blockchain se considera una de las más seguras, a costa de necesitar una gran carga computacional para funcionar.

### *2.1.2. Tipos de red: públicas y privadas*

En blockchain se distinguen dos tipos de redes, una publica, tal y como puede ser la red de Ethereum, accesible por cualquiera que disponga de una copia de ella y una cartera, y en la que todos los movimientos dentro de la red son públicos para todo el mundo; y otra privada, por norma general creadas por entidades privadas para aprovechar su funcionamiento en un entorno reducido, en las que la transparencia de la red puede ser publica o no, pero principalmente se restringe la escritura de datos en la red a una lista de usuarios limitada por la propia entidad.

La mayoría de las redes públicas funcionan gracias a las criptomonedas, dado que por defecto trabajan en una red basada en la prueba de trabajo, requiriendo de una cantidad enorme de usuarios que proporcione potencia computacional para trabajar en los problemas criptográficos a la hora de crear nuevos bloques, y así recompensando a los usuarios con una moneda digital a cambio de esa fuente de energía.

Por otra parte, la ventaja de las redes privadas es que eliminan esa necesidad de potencia computacional dado que si reducimos el número de nodos a un centenar respecto a los millones

que puede suponer una red pública, es mucho más sencillo organizar y poner de acuerdo con todos los nodos cada vez que se registre una transacción en la red.

Las aplicaciones más comunes en este tipo de redes suelen ser una base de datos administrada por los empleados de la propia entidad o un sistema de notaría para la empresa sin necesidad de utilizar una red pública, y en la que solamente aquellos autorizados sean capaces de obtener los datos de la red cuando se necesiten. Esto reduce enormemente los costes debido a que no hay que pagar a los mineros de la red pública para que se verifiquen las transacciones, pero a su vez elimina la descentralización ya que todo se verifica desde la misma organización que la maneja, relegando una de sus mayores virtudes a otro plano.

## 2.2. Minado de bloques, Proof of Work y Proof of Stake

Cuando se habla de las redes tipo *Proof of Work* (PoW) o *Proof of Stake* (PoS) se refieren al tipo de sistema o mecanismo interno que tiene la red para minar un bloque y validar la transacción consiguiendo así evitar ataques maliciosos que puedan alterar el estado de la blockchain. Gran parte de las criptomonedas utilizan una red con el mecanismo de prueba de trabajo y la minería es el foco central de la red.

El minado de bloques es el nombre que se le da al proceso que conlleva crear un nuevo bloque. Normalmente, cuando se habla de minería, se habla sobre el sistema de prueba de trabajo que podemos ver en la figura 2. Para minar un bloque, los mineros ponen a trabajar a sus nodos para realizar un problema matemático dado por la red que requiere de una gran potencia computacional. Conforme más grande se vuelve la red, más cuesta minar en ella por lo que en redes tan grandes como la de Bitcoin o Ethereum sería imposible para un solo ordenador resolver el problema y minar el bloque. Estos problemas suelen estar relacionados con la criptografía, por lo que se necesita de una gran cantidad de dispositivos que estén activos y calculen posibles soluciones de forma cooperativa. Así, aunque no sea tu dispositivo el que termine de minar un bloque, aquellos usuarios que presten su potencia a la red acaban validando el bloque que ha sido minado, siendo así recompensados en relación con el trabajo proporcionado.

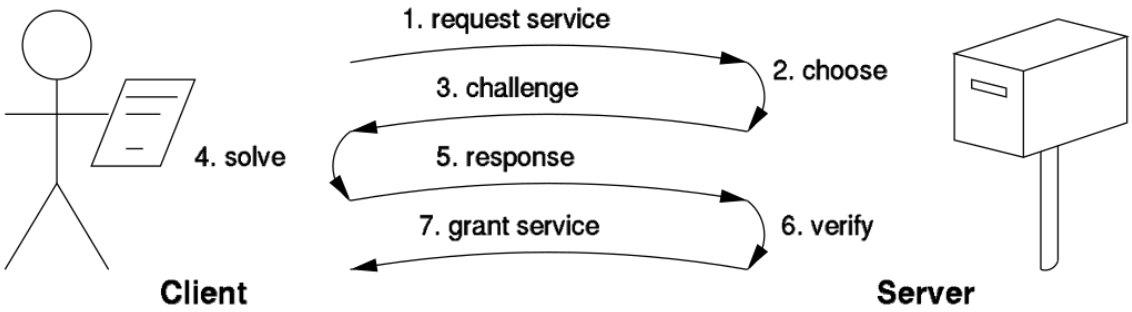


*Figura 2 – Sistema de prueba de trabajo. Fuente: https://en.wikipedia.org/wiki/Proof_of_work*

Este proceso se divide en varias etapas [2]: La primera es la transacción, la cual es el comienzo del minado del nuevo bloque. Esta depende de que un usuario realice una transacción en la red, ya sea un intercambio de dinero entre dos usuarios o una interacción con un contrato inteligente.

Luego viene la compilación, en la que las transacciones se agrupan en un bloque que será utilizado por los mineros para empezar a trabajar.

Después llega la formación, donde las transacciones seleccionadas por cada minero formaran un bloque que será candidato para ser el siguiente bloque de la cadena. Este contendrá el hash del anterior bloque, metadatos como la marca de tiempo y el contenido de las transacciones.

Tras esta, se pasa a la prueba de trabajo. En ella, los mineros deben encontrar un hash válido para su nuevo bloque, realizando un cálculo matemático único por cada bloque que han formado. Puede ocurrir que varios mineros dispongan de transacciones repetidas en un bloque anterior, pero al ser validadas se eliminan del nuevo bloque sin ningún problema. La solución que deben tomar los nodos mineros para encontrar el hash se basa en repetir cálculos junto a un *nonce* o '*number once*', siendo este un número arbitrario que se usa una única vez y que va cambiando constantemente hasta que se halle un hash válido según la condición de la red para el siguiente bloque. Como no se puede predecir que *nonce* es el que resuelve el problema criptográfico, se necesita una gran cantidad de nodos mineros para hallar la solución final.

Una vez un minero encuentra un hash válido, entramos en la etapa de transmisión, donde el resto de los nodos mineros reciben el bloque con el hash para poder validarlo. El nodo que lo ha encontrado recibe una recompensa en base al premio establecido por la red en valor de criptomonedas (en Bitcoin se recompensa 6.25 bitcoins por bloque minado, aproximadamente

240.000€) junto a las comisiones de minado de todas las transacciones que incluya ese bloque. Estas comisiones son pagadas por los usuarios como una cuota por el servicio de los mineros y van cambiando en base a la utilización de la red.

La recompensa se le da únicamente al nodo minero que encuentra el hash, por lo que a no ser que un usuario tenga una gran cantidad de nodos minando a la vez es bastante improbable que consiga minar un bloque antes que el resto. Así nacieron las piscinas de mineros o *mining* pools. Estas son una agrupación de mineros que ponen a trabajar sus nodos de forma cooperativa de modo que miles de usuarios participan en la red como si fueran un solo ente, poniendo a la disposición del grupo su potencia computacional y así logrando una rapidez de minado mucho mayor y utilizando menos recursos energéticos. Si una de estas piscinas consigue minar un bloque, la recompensa se distribuye entre todos los nodos que hay en ella en base al trabajo realizado por cada uno. Esta es la forma más común de participar para obtener una recompensa y lo que se considera como estándar en minería.

Después de la transmisión, queda que el resto de los nodos validen el bloque en la etapa de verificación. Los nodos que participan en el minado de la red verifican que el bloque no tiene manipulación alguna y que el hash es válido en base a como está establecido en el blockchain. Una vez validado, la prueba de trabajo del minero se valida también y el minero puede pasar a utilizar las criptomonedas.

Finalmente, cuando ya se ha verificado el bloque, solo queda confirmarlo. Los bloques que vayan siendo añadidos a la red confirmarán que el bloque es válido, y el resto de los usuarios que estaban minando el mismo bloque deberán desechar su trabajo ya que el último hash válido es el del nuevo bloque.

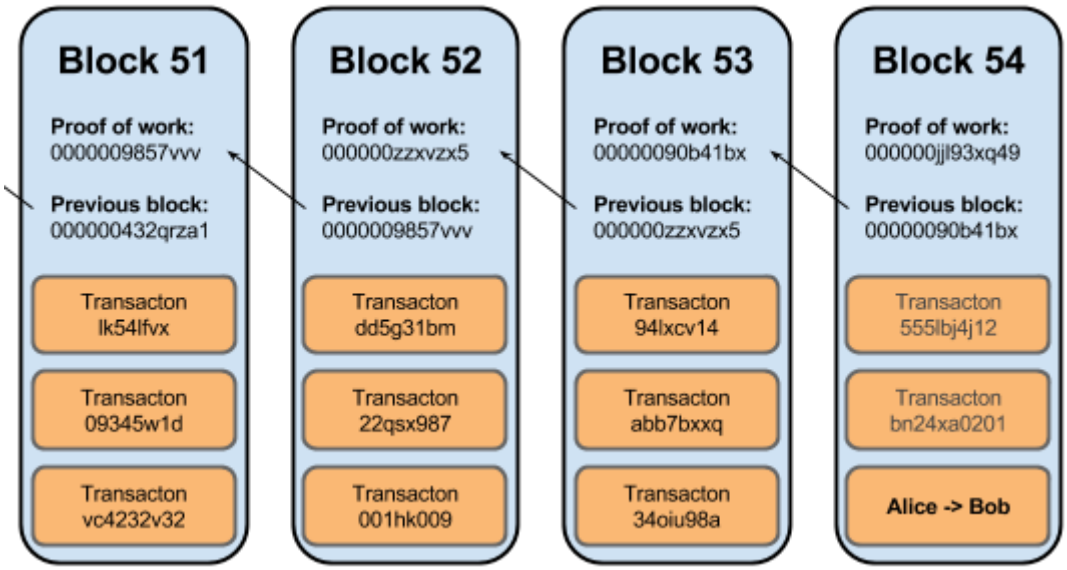


*Figura 3 - Esquema de bloques en una red de blockchain. Fuente: https://moften.medium.com/qu%C3%A9-es-el-blockchain-y-c%C3%B3mo-se-usa-para-combatir-el-fraude-b4e480446b2d*

Todo este proceso de minado se basa en una red *Proof of Work*, pero existe otra alternativa.

El sistema de *Proof of Stake* o prueba de participación se basa en eliminar la competitividad de minería de la red a través de un sistema que verifique los bloques en base a la participación de los nodos, pero sin ser necesario utilizar una gran potencia computacional, creando un consenso entre todas las partes integrantes de la red de blockchain. [3]

Los nodos mineros son llamados validadores y la validación del bloque se asigna entre los nodos de forma aleatoria en base a ciertos requisitos, entre ellos la cantidad de monedas que tenga el usuario y el tiempo que han estado en funcionamiento. La probabilidad de que el nodo de un usuario sea elegido será altamente influenciada por esos factores, pero debido a que es aleatorio, puede ocurrir que un nodo con pocos recursos y poco tiempo en la red sea elegido igualmente para validar un bloque.

Mientras que en el mecanismo de prueba de trabajo se requiere de hacer un trabajo computacional elevado para validar un bloque, en las redes PoS la dificultad de ese trabajo se modifica proporcionalmente respecto a la cantidad de monedas que posee el minero, y en algunas monedas como Peercoin, se añade a la formula el tiempo que ha mantenido esas monedas en su posesión, requiriendo así una carga energética mucho menor y consiguiendo un modelo más amigable con el medio ambiente. Se estimaba que, en 2015, una transacción de

Bitcoin necesitaba la misma cantidad de energía que 1.57 casas estadounidenses al día, y ese número no ha hecho más que crecer. [4]

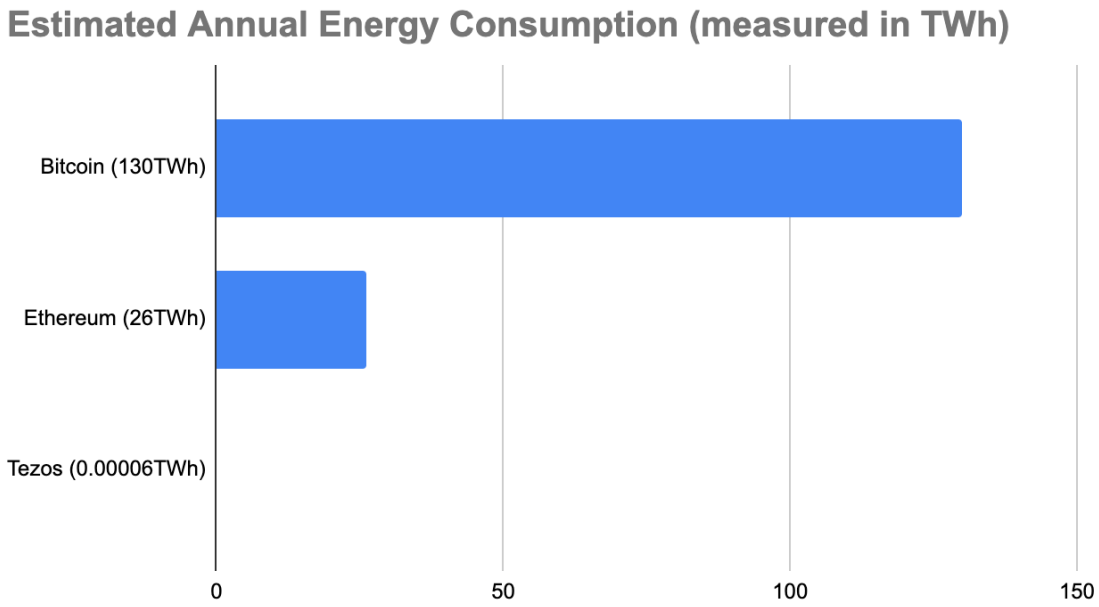


*Figura 4 - Gasto anual en TWh de Tezos (PoS) respecto ETH y BTC. Fuente: https://medium.com/tqtezos/proof-of-work-vs-proof-of-stake-the-ecological-footprint-c58029faee44*

Una de sus grandes ventajas es que solventa el llamado "ataque del 51%" el cual se basa en que, si un grupo de usuarios fuese capaz de controlar el 51% de la red, serían capaces de hacer lo que quisieran con ella, dado que la red se distribuye usando un protocolo consensuado entre los participantes. En redes tan grandes como Bitcoin sería muy complicado poder lograrlo, pero este problema florece en redes pequeñas en las que con el suficiente poder computacional podrían realizarse ataques de denegación de servicio (DoS) que afectarían al resto de usuarios.

Las redes PoS solventan este problema con su característica fundamental, siendo que para que un grupo pudiese realizar un ataque sobre la red necesitarían disponer del 51% de las monedas en circulación, algo que es bastante poco probable siendo que puede haber miles de millones de usuarios entre los que se distribuyan esas monedas. A parte, con esa gran cantidad de posesión, el valor de la moneda bajaría y provocaría grandes pérdidas económicas para aquellos que intenten atacar la red, creando así un sistema disuasorio para este tipo de ataques.

La red de Ethereum está en proceso de cambio hacia una red basada en prueba de participación para obtener una mayor descentralización, ser más eficiente energéticamente y reducir la potencia de hardware necesario para participar en ella. Siendo la segunda red más grande, supone un gran cambio para la industria del blockchain y cómo se aplicarán sus tecnologías desde ese momento en adelante.

## 2.3. Smart Contracts

Los contratos inteligentes suponen la evolución de las redes de blockchain como algo más que un listado de transacciones entre usuarios, permitiendo a los desarrolladores crear aplicaciones basadas en la lógica interna de la red y expandiendo las posibilidades de crear soluciones a problemas existentes.

Su función principal es automatizar un proceso lógico entre el usuario que realiza la transacción y el destinatario, permitiendo ejecutar un código específico en el momento que se realiza esa transacción dentro de la red. En Ethereum, los contratos inteligentes tienen su propia dirección, al igual que una cartera, y pueden recibir como enviar criptomonedas.

Hay otras redes de blockchain que permiten usar contratos inteligentes como Ripple (XRP) o incluso Bitcoin de forma muy primitiva, pero dado que el proyecto está basado en Ethereum, se hablará de los contratos inteligentes de dicha red. [5]

En Ethereum, los *Smart Contracts* funcionan en una máquina virtual llamada EVM (Ethereum Virtual Machine) y son programados en el lenguaje Solidity. Su funcionamiento simula a un contrato automatizado que elimina la parte del contratista, registrando en el blockchain dicha transacción del contrato y permitiendo gestionar lo que ocurra después de ese intercambio. Por ejemplo, se podría utilizar un contrato inteligente para comprar un piso cuya cerradura funcionase por un código. Al mandar el pago a la dirección, el propietario de la casa cambiaria automáticamente a la persona que realiza el pago sin necesidad de un intermediario, y esta quedaría registrada de forma pública en la red, de modo que se podría verificar en cualquier momento que el nuevo propietario es el verdadero. Una vez pagada, se mandaría un código de acceso a la casa para el nuevo dueño, por lo que no sería necesario ir a una agencia de pisos para recibir las llaves y el propietario podría acceder directamente a su nueva vivienda. [6]

Estos *Smart Contracts* permiten simplificar trámites burocráticos y la necesidad de rellenar documentos en papel. Simplemente, con la transacción y la firma digital que aporta una dirección única en el blockchain bastaría.

Aunque actualmente no existe una vasta implementación de los contratos inteligentes, posiblemente en un futuro haya una gran cantidad de tecnologías y aplicaciones que hagan uso de estos y de la red de blockchain debido al gran apoyo que puede aportar a sectores como el sector inmobiliario, el sector sanitario, comercios, bancos, contratos para empresas, procesos electorales y mucho más.

En el apartado de desarrollo, se explicará más a fondo la tecnología detrás de los *Smart Contracts* y Solidity.

## 2.4. Aplicaciones ya existentes

Debido a la popularidad de los contratos inteligentes, es normal que existan aplicaciones que implementen un sistema de notaría digital ya que por las características del blockchain resulta ser una de las ideas más atractivas como posible negocio. Dos de las aplicaciones que implementan este sistema son NotariSed y Blocknotary.

Estas aplicaciones tienen un equipo de desarrollo detrás y una gran inversión por parte de una empresa que les permite disponer de mayores recursos a la hora de desarrollar el software.

NotariSed, funciona gracias a su propia criptomoneda que permite a los usuarios un control mayor del pago a la hora de su utilización y ofrece un gran set de servicios para el cliente y su seguridad.

Luego Blocknotary, divide su aplicación en distintos productos que van desde la notaría de documentos hasta entrevistas verificadas por la red de blockchain.

Siendo estas aplicaciones los grandes competidores en el apartado de notaría digital, ambas páginas requieren del registro y de la compra o pago por estos productos. En el caso de NotariSed se debe comprar un token que debe ser convertido de otra criptomoneda, cuyo proceso puede resultar confuso para el cliente, y requiere que el usuario registre su cuenta almacenando esta en su base de datos. Blocknotary sin embargo utiliza un sistema de separar

su aplicación por productos haciendo que el usuario tenga que descargar varias aplicaciones según sus requisitos.

Para este proyecto, se decidió que no fuese necesario crear una cuenta dentro de la aplicación, facilitando así un uso instantáneo únicamente descargando una extensión en el navegador. Además, los fondos utilizados para pagar los requisitos de la aplicación son los Ether, una criptomoneda completamente segura y funcional bajo la red de Ethereum que puede ser adquirida en cualquier servicio de intercambio sin problema alguno. Este proyecto también nace con la idea de que uno de los modelos de negocio sea personalizar el producto para otra empresa.

# 3. Objetivos

Los objetivos del proyecto se establecieron en la propuesta que se puede encontrar en el anexo. Estos no han cambiado puesto que se ha seguido el planteamiento inicial y se ha desarrollado en base a los mismos.

Tal y como se define en el documento de la propuesta, los objetivos del proyecto son:

1. Analizar las diferentes tecnologías Blockchain existentes y su integración con sistemas actuales de identidad digital.
2. Definir y desarrollar la arquitectura necesaria para el uso del sistema.
3. Diseñar y desarrollar los APIs e interfaces básicos para el manejo e integración del sistema.
4. Analizar las posibles oportunidades y modelos de negocio derivados.

# 4. Metodología

La metodología del proyecto se estableció en la primera reunión con la empresa Inycom (se puede encontrar el acta en el anexo, página X).

## 4.1. Elección de metodología

Principalmente se discutió qué metodología debía seguir. Para que el proyecto tuviese un desarrollo adecuado concluimos que era mejor seguir una metodología de desarrollo ágil de software como SCRUM, pero decidimos que lo adaptaríamos a nuestra manera para hacerlo más sencillo.

Al final, decidimos hacer un sistema de reuniones, similar a los *sprints*. Cada dos semanas aproximadamente se planearía una reunión, exceptuando algunos parones por el curso de la universidad. En estas, se mostrarían las tareas realizadas, en caso de haber problemas con el desarrollo, estos se analizarían y se dedicaría la reunión a solventarlos en la medida de lo posible, y finalmente se asignarían nuevas tareas para la siguiente reunión. Así el proyecto progresaría de forma adecuada, adaptando las necesidades en base a los resultados de cada reunión.

## 4.2. Evolución del proyecto

El desarrollo del proyecto se basó en tres fases divididas para la implementación de este.

La primera fase fue planificada como una fase de aprendizaje y documentación, en la que debía entender y saber explicar cómo funcionan las tecnologías de blockchain, tal y como se han explicado en el apartado del estado del arte. Entre ellas, debía entender las redes de prueba de trabajo, las redes de prueba de participación, y las carteras digitales. Después, dejar claro con qué tecnología quería trabajar la empresa.

Una vez entendido eso y decididas las herramientas que se iban a utilizar se pasaría al desarrollo y la implementación. Esta trataría de un cliente básico que integrase las funcionalidades que se requerían, lo cual necesario para comprender correctamente como interactuar con la red de Ethereum y poder crear una aplicación final.

Después de tener todas las funcionalidades listas, la fase final se centraría en adaptar todas esas funcionalidades a una página web real en un estado base y que en un futuro pudiese readaptarse para un cliente de la empresa.

Finalmente, una vez estuviese implementado todo, se dedicaría el resto del tiempo al desarrollo de la memoria. Durante las reuniones y la implementación se fueron realizando apuntes sobre qué se iba explicando, implementando y los cambios conforme avanzaba el desarrollo para luego plasmarlos en la memoria.

### 4.3. Tareas, tiempo y reuniones

Según la planificación que se fue definiendo en las reuniones, se puede sacar un listado de tareas que se debían realizar para la implementación del proyecto y el desarrollo de la memoria.

Para el apartado de implementación y documentación:

- Aprender sobre la tecnología de blockchain.
- Crear una red local de Ethereum con Ganache.
- Aprender sobre Solidity y crear un *Smart Contract* básico para testear en la red local.
- Desarrollar un *Smart Contract* más avanzado y lanzarlo en la red pública.
- Subir documentos a la red con IPFS.
- Crear un cliente web básico que interactúe con el *Smart Contract* utilizando Web3JS y haga llamadas a la red con un nodo de Infura.
- Publicar el *Smart Contract* definitivo y desarrollar la página web.
- Optimizar código y tener una demo lista para la presentación.

Las tareas del desarrollo de la memoria se pueden resumir en documentar lo aprendido, buscar ejemplos que puedan ser entendidos por alguien sin conocimientos previos, buscar imágenes que ilustren lo que se quiere explicar y escribir cada apartado.

En cuanto al tiempo requerido, este se puede ver en la tabla de la figura 5 con los meses correspondientes, el tiempo invertido y las tareas realizadas.

| | NOVIEMBRE | DICIEMBRE | MAYO | JUNIO | JULIO | AGOSTO | SEPTIEMBRE |
|---|---|---|---|---|---|---|---|
| ***APRENDIZAJE SOBRE BLOCKCHAIN*** | 4 DIAS | 1 DÍA | | | | | |
| ***RED LOCAL DE ETHERUM FUNCIONAL*** | | 1 DÍA | 2 DIAS | | | | |
| ***IMPLEMENTACIÓN EN LA RED PÚBLICA*** | | | 3 DIAS | 1 DÍA | | | |
| ***CREACION DEL CLIENTE DUMMY*** | | | 1 DÍA | 10 DÍAS | 2 DÍAS | | |
| ***DESARROLLO DE LA APLICACION WEB*** | | | | | 5 DÍAS | 1 DÍA | |
| ***DESARROLLO DE LA MEMORIA*** | | | | | 1 DÍA | 15 DÍAS | REVISIÓN 5 DIAS |

| | |
|---|---|
| ***TIEMPO DEL DESARROLLO*** | **186 HORAS** |
| ***TIEMPO DE LA MEMORIA*** | **126 HORAS** |

**DÍA = 6 HORAS**

**TOTAL = 312 HORAS**

*Figura 5 - Tabla de tiempo invertido en el proyecto.*

# 5. Desarrollo

En este apartado de desarrollo se va a tratar las diferentes secciones que han compuesto la implementación del proyecto, desde el estudio de las tecnologías de blockchain hasta la creación de la página web definitiva.

Se ha separado en 6 secciones, siendo la primera una justificación del uso de las tecnologías y las herramientas, y el resto los hitos más importantes del desarrollo del proyecto.

## 5.1. Tecnologías y herramientas

Para realizar la implementación del proyecto se decidió utilizar Ethereum como tecnología de blockchain. Las razones se dividen en una razón económica que se explica en el estudio económico y una razón técnica que se más tarde en este apartado. A parte de estas, una de las razones de peso es la cantidad de información que hay en internet sobre Ethereum y la gran comunidad que tiene detrás. Al ser la criptomoneda que popularizó los contratos inteligentes mucha gente empezó a desarrollar aplicaciones en Ethereum y hoy en día es la que más soporte recibe.

Solidity viene de la mano de los contratos inteligentes pues es el lenguaje de programación utilizado para desarrollar Smart *Contracts*. Este puede ser ejecutado en la nube gracias a un entorno de desarrollo integrado (IDE) llamado Remix. En él, puedes compilar tu *Smart Contract* en cualquier versión publicada de Solidity, interaccionar con el contrato y ver su funcionamiento en vivo, y publicarlo en la red local o en una pública de pruebas para darle un uso real.

Para el apartado web se pensó en utilizar TypeScript como lenguaje, pero debido a la falta de conocimientos sobre el lenguaje la decisión final ha sido usar JavaScript.

Respecto al servidor de la web, se ha utilizado NodeJS, el cual funciona como un entorno de ejecución de JavaScript y me ha ayudado a tener la web en un entorno local.

El *front-end* de la página funciona con React, una librería de JavaScript que permite el desarrollo de aplicaciones en una sola página, trabajando con datos que cambian constantemente. Este permite crear componentes que tienen su propio estado y así manejar datos en tiempo real sin necesidad de recargarla.

Para poder conectar la aplicación a la red de Rinkeby, una red pública de Ethereum para pruebas en la que puedes funcionar con criptomonedas ilimitadas a modo de aprendizaje, ha sido necesario un nodo proporcionado por Infura. Una vez obtenido dicho nodo, se han utilizado las librerías Web3JS y JS-IPFS para interactuar con él. Web3JS se encarga de conectar con la red, conectar con una cartera digital y poder mandar peticiones. Luego JS-IPFS se encarga de conectar con los nodos de IPFS y cargar un archivo el cual será devuelto como una cadena de caracteres, los cuales sirven como identificador para el documento dentro de la red IPFS. Esta funciona como una red P2P entre los nodos, haciendo así que subir un documento en la red de Ethereum sea muchísimo más barato.

Como herramienta de desarrollo para la web se ha utilizado Visual Studio Code, un IDE ligero y amplio en extensiones que permite personalizar la vista del editor de texto y a su vez brinda la posibilidad de compilar con consola desde el propio entorno.

Finalmente, GitHub ha servido como sistema de repositorio en el que se ha acabado subiendo la página web finalizada.

## 5.2. Documentación y estudio sobre Blockchain

En esta sección se profundiza un poco más sobre la parte del estudio para la implementación del código, ya que, aunque se haya dedicado el mismo tiempo a aprender sobre el blockchain en términos generales (apartado 2.1 del estado del arte), en esta sección deja de ser relevante, por lo que sólo se hablará sobre la parte técnica que sea relevante para entender el código.

### *5.2.1. Transacciones en la red*

Toda interacción con la red queda registrada en una transacción. Para enviar una transacción a la red se necesita un nodo que esté conectado y disponga de una copia del blockchain. Estas transacciones se dividen en distintos datos como la fecha en la que fue realizada o la dirección de la cartera que la realiza y la dirección de a donde se dirige entre otros.

Es importante entender los datos que representan una transacción dado que es la parte fundamental de la red y lo que permite la trazabilidad de la información que se suba, por lo que se presenta un desglose de cada dato que se puede ver en las figuras 6, 7 y 11.

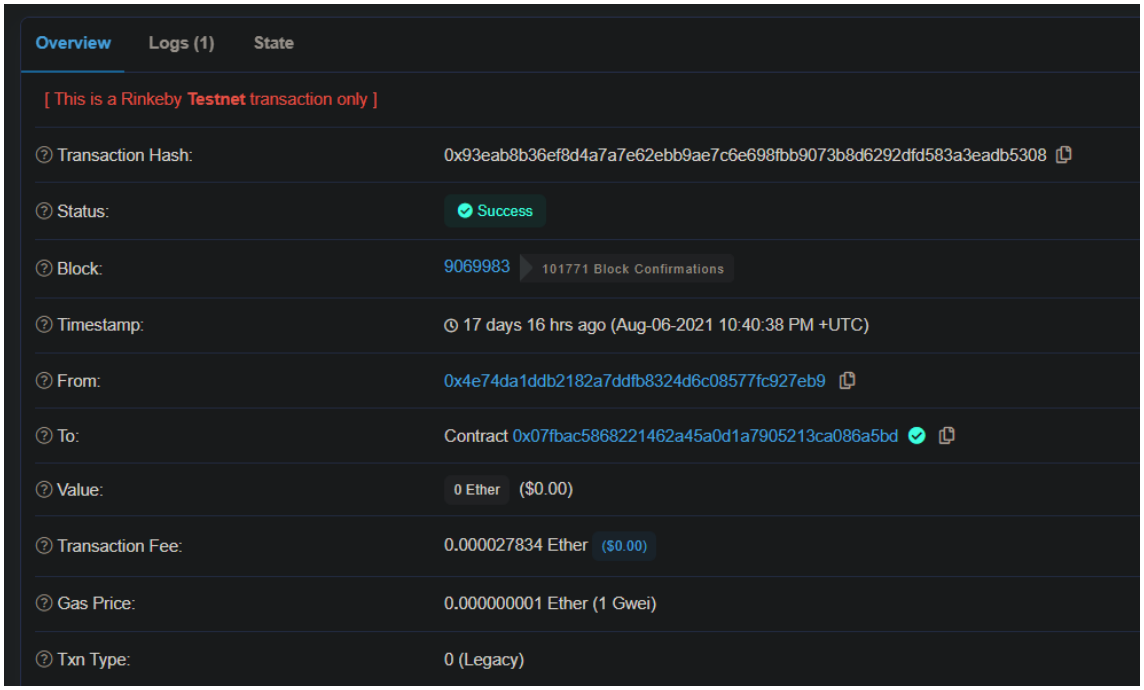


*Figura 6 - Ejemplo de transacción realizada a mi contrato inteligente.*

Gracias a un explorador de bloques como Etherscan se pueden analizar las transacciones realizadas en la red. Esta es una transacción realizada al *Smart Contract* del proyecto en la que se almacena el hash encriptado con el identificador del documento y varios metadatos.

Primero, los datos principales que tienen todas las transacciones son:

- *Transaction Hash*: Es el identificador único que tiene esa transacción en toda la red, el cual permite localizarlo en páginas exploradoras de bloques.
- *Status*: El estado actual en el que se encuentra esa transacción. Puede ser *Success*, siendo una transacción validada, o *Failed*, en caso de que haya habido algún problema, devolviendo al usuario la cuota de la transacción.
- *Block*: Indica el número del bloque al que pertenece la transacción. El número que aparece junto al número de bloque se denomina *Block Confirmations* o confirmaciones de bloque, el cual representa la cantidad de bloques que han sido creados a partir del bloque en el que se encuentra dicha transacción. A mayor sea ese número, más segura se vuelve la validez de la transacción.
- *Timestamp*: Muestra la fecha exacta en la que se creó esa transacción en la red.
- *From*: Dirección de la cartera que actúa como emisor de la transacción.
- *To*: Dirección de la cartera o contrato inteligente que funciona como receptor de la transacción.

- *Transaction Fee*: Coste en Ether (la criptomoneda de Ethereum) de lo que ha costado publicar esa transacción en la red. Este coste varía en base a la utilización de la red y la cantidad de datos que se tengan que almacenar.
- *Gas Price*: El gas es "una unidad que mide la cantidad de esfuerzo computacional requerida para ejecutar operaciones específicas en la red de Ethereum" [7]. El precio suele estar expresado en Gwei, donde una sola unidad de Gwei equivale a $10^{-9}$ Ethers. Este precio supone una comisión para el usuario debido al coste computacional requerido por los mineros para procesar su transacción.
- *Txn Type*: Tipo de la transacción. Implementado en la revisión de Ethereum EIP2718. A todas las transacciones se les asigna el tipo *Legacy* (0) ya que es un valor implementado para nuevos tipos de transacciones en el futuro.

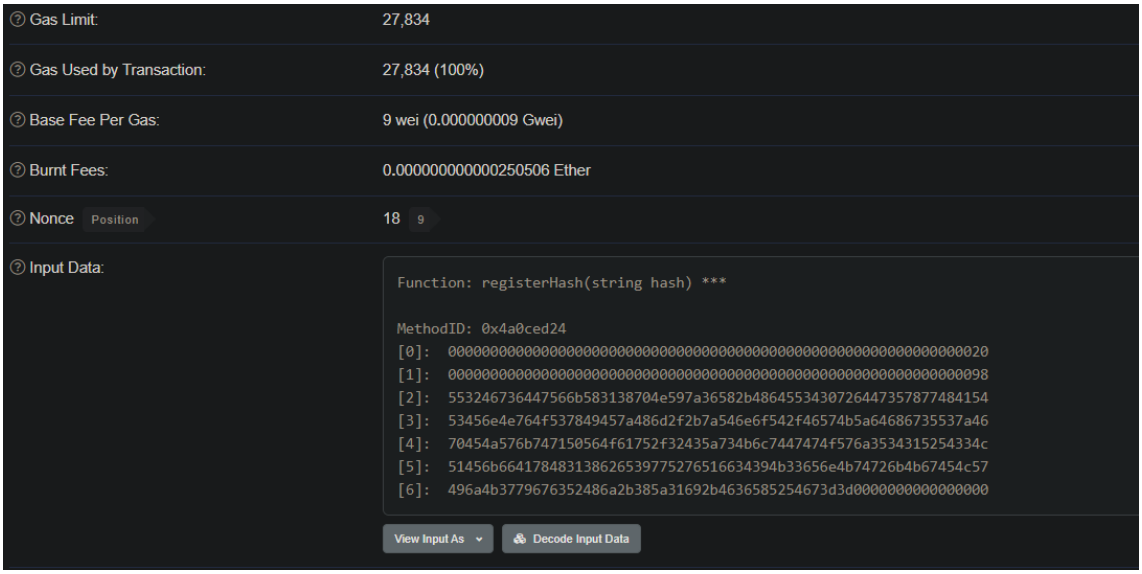


*Figura 7 – Resto de datos de una transacción.*

Luego, hay unos datos extra que pueden ser relevantes para usuarios más avanzados como se pueden observar en la figura 7:

- *Gas Limit*: El límite de gas se establece a la hora de mandar una transacción. Este puede ser personalizado por el usuario como podemos comprobar en la figura 8, y supone el límite de gas que estás dispuesto a utilizar para esa transacción. Comúnmente al ir a crear una transacción el *Gas Limit* se asigna en base a la red por la aplicación utilizada, ya que a mayor sea el número, mayor coste supone para el usuario. En caso de que ese límite sea superado debido al coste computacional requerido para procesarla, esta será invalidada y su estado pasara a *Failed*.

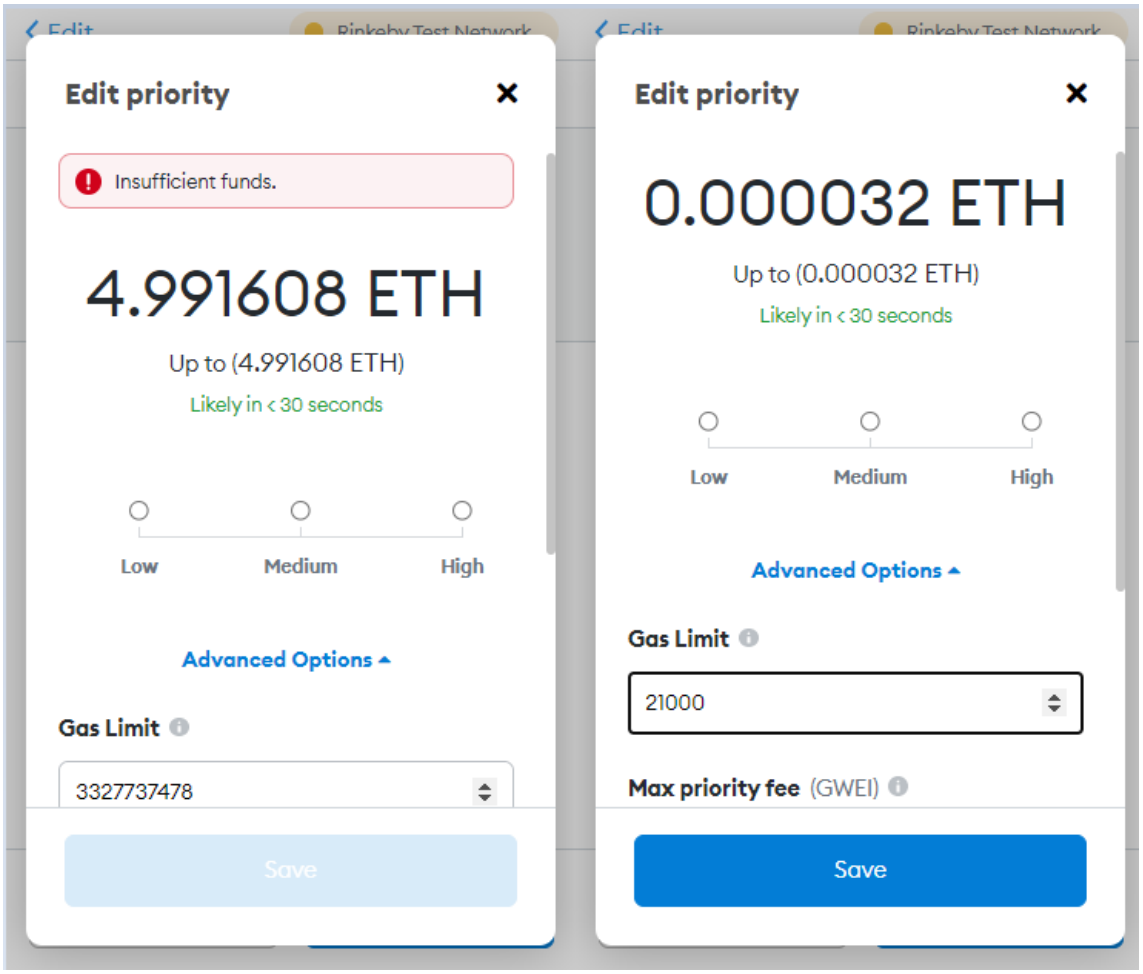


*Figura 8 - Comparativa de la personalización del límite de gas en una transacción a través de Metamask.*

- *Gas Used by Transaction*: La cantidad de gas que se ha utilizado finalmente para procesar la transacción. Puede ser menor que el límite establecido, pero por norma general se especifica en la aplicación el límite exacto requerido para ahorrar costes.
- *Base Fee per Gas*: Después de la actualización London de Ethereum, se incluyó un coste base en los bloques que representa el coste mínimo por unidad de gas para incluir esta transacción en el bloque. Además, a este coste se le puede añadir una propina que permite acelerar el proceso de inclusión, haciendo que los mineros prioricen la transacción a más alta sea la propina.
- *Burnt Fees*: El valor de las tasas base por unidad de gas se elimina de la red, es decir, no van a ningún destinatario. Solamente la propina será la recibida por el minero que mine el bloque. Esto es debido a la actualización London de la red que incluyó un protocolo para comenzar a aumentar el quemado de Ether, uno de los principios básicos de Ethereum como se puede ver en la figura 9.

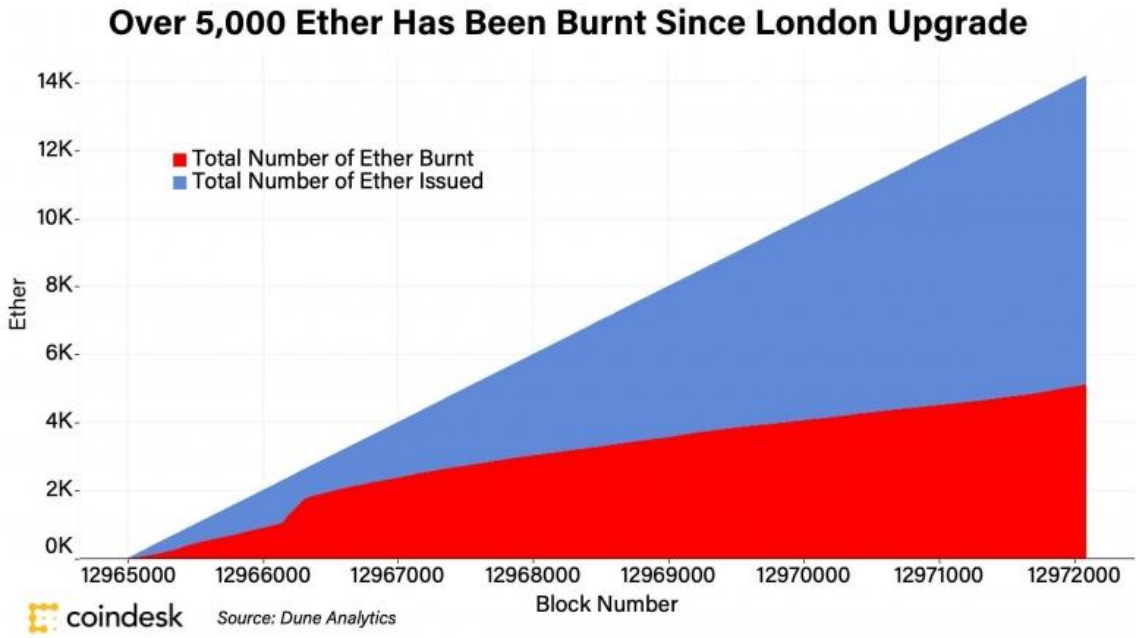


*Figura 9 - Ether quemado en base al número de Ether producido. Fuente: https://www.coindesk.com/ethereum-burns-5000-coins-in-2-days*

- *Nonce*: Distinto al *nonce* que se explicaba en el minado de bloques, este es un número secuencial que representa el número de transacciones que han sido realizadas por el emisor. Comienza en 0 para la primera transacción, de modo que como se aprecia en la figura 7, esa será la decimonovena transacción del usuario. El número que aparece a su lado representa la posición de esa transacción en el bloque.
- *Input Data*: Información adicional que se puede incluir en una transacción. Por defecto, no se incluye nada en ella, pero aquí es donde se representan los datos con los que interactúan los contratos inteligentes. Estos datos son representados de forma hexadecimal y solo pueden ser decodificados si se dispone del código del *Smart Contract* o este es público en la red. En la figura 10 se aprecia que los datos que almacenan esta transacción son una cadena de caracteres.


*Figura 10 - Datos decodificados de la transacción.*

Finalmente, se encuentra la parte más importante de la transacción para lo que ha sido trabajado en este proyecto. Debido a cómo funciona el contrato inteligente que he creado, hay un tipo de dato especial que se almacena en las transacciones, y este es el registro de eventos o Logs. [8]

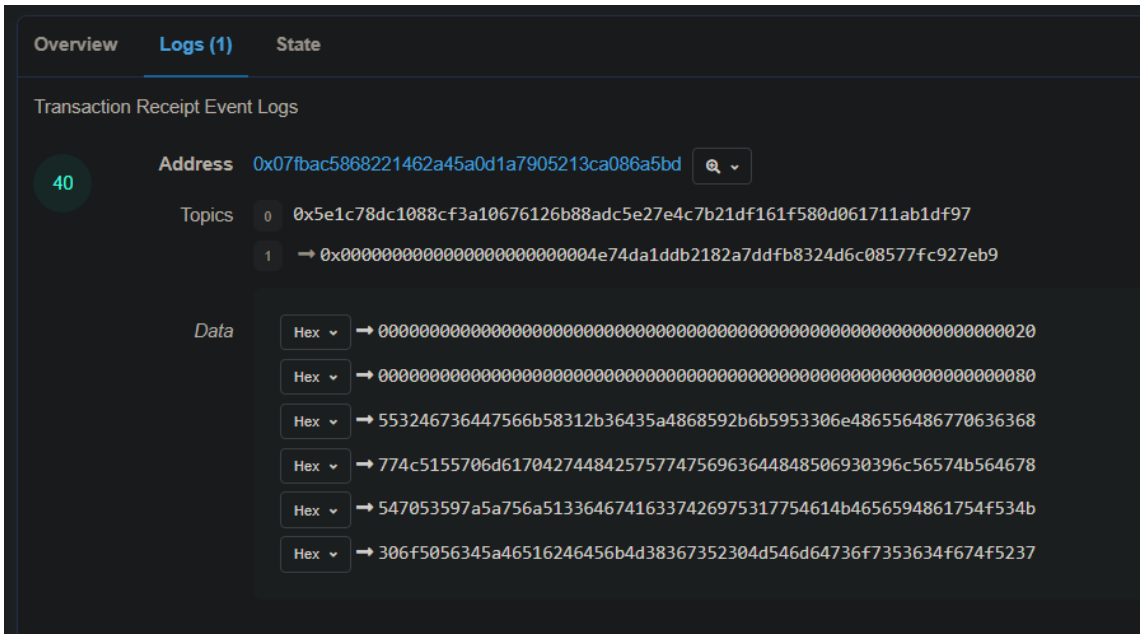


*Figura 11 - Logs de eventos de la transacción.*

Estos registros almacenan los eventos que han sido emitidos por el contrato inteligente. Los registros están compuestos por los temas y los datos. Los primeros son cadenas de 32 bytes que definen la utilidad de ese evento y están limitados a 4 temas por registro. El primer tema siempre es el nombre del evento que ha sido emitido, mientras que el usuario puede añadir hasta 3 temas más. Los temas permiten buscar esos eventos en los bloques de la red haciendo de los registros de eventos una funcionalidad extremadamente útil.

La segunda parte son los datos que almacenan ese evento. En el caso del proyecto, los datos almacenados componen el hash encriptado del documento y los metadatos que se han subido. La ventaja de los datos es que no tienen límite de bytes como los temas, pero al contrario de estos, los datos no tienen la capacidad de ser buscados por lo que es importante definir correctamente un tema para los eventos y así poder acceder a los datos desde el código.

Para crear uno de estos eventos es necesario un contrato inteligente que tenga una función de emisión de eventos, los cuales se van a explicar en el apartado 5.2.2 sobre Solidity.

### *5.2.2. Solidity*

Solidity es el lenguaje de programación utilizado por los contratos inteligentes en la red de Ethereum que permite aplicar una lógica a las transacciones.

El código comparte una sintaxis similar a JavaScript, pero requiere tipado en las variables. Este se ejecuta en la Ethereum Virtual Machine de un nodo, la cual interpreta el código y lo compila para traducirlo a instrucciones en código de bajo nivel.

```
pragma solidity >= 0.7.0 <0.8.0;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent (address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() public {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent (msg.sender, receiver, amount);
    }
}
```

*Figura 12 - Ejemplo de código en Solidity. Fuente: https://es.wikipedia.org/wiki/Solidity*

Como se puede ver en el ejemplo de la figura 12, los contratos son similares a una clase en otros lenguajes. Estos pueden almacenar variables y funciones complejas que puedan ser llamadas desde el cliente.

La primera línea de código que tiene que haber en un código de Solidity siempre es el *pragma solidity*. El *pragma* es una palabra reservada que especifica con qué versión del compilador se va a compilar ese código. Dado que Solidity está en continua evolución, esto permite a los desarrolladores elegir sobre que versión quieren trabajar sin necesidad de cambiar todo el código a la última versión.

A parte del pragma, las palabras reservadas del lenguaje más relevantes para el proyecto son las *address* o direcciones y los *events* o eventos.

Las *address* son un tipo de variable creado específicamente para almacenar una dirección de una cartera de Ethereum. Estos son representados por 20 bytes y una de sus propiedades es que pueden ser de tipo *payable*, es decir, una dirección a la que se le puede enviar dinero.

La palabra reservada de *payable* también funciona como un modificador para funciones, haciendo un requisito de pago para realizar una llamada a la misma.

Luego los eventos son un miembro heredable de los contratos que permiten almacenar datos en los registros de las transacciones. Esto permite que los datos que se mandan al llamar a un evento no se queden guardados en una variable del contrato, por lo que, si se crea un contrato con únicamente eventos, se puede conseguir un contrato sin estado.

Los contratos por norma general tienen un estado en base a las variables que tengan. Por ejemplo, si al registrar el contrato hay un método constructor que asigne la cartera que lo ha creado a esa variable, se puede decir que ese contrato es *stateful*, y esa variable será registrada en los datos del contrato. A más datos contenga un contrato, mayor es su peso en la red, por lo que realizar transacciones que modifiquen su estado actual se convierte en algo más costoso.

Es ahí donde entra en relevancia los eventos de Solidity. Los eventos permiten hacer llamadas a la función que los emite y hacer que eso se registre en la transacción en vez de los datos del contrato, consiguiendo así que las llamadas sean más baratas en cuanto a precio de gas. Esto

se explicará más a fondo en la decisión sobre cómo se creó el *Smart Contract* y en el estudio económico a la hora de calcular el precio que debería pagar el cliente por utilizar el servicio.

## 5.3. Nodo de Ethereum local y primer Smart Contract

Después del aprendizaje sobre Solidity y las transacciones en la red de Ethereum, el siguiente paso fue el desarrollo de un *Smart Contract* básico capaz de almacenar algún dato y con una función *setter* y otra función *getter*.

Luego, este contrato debería ser lanzado en una red local de Ethereum. Se recomendó utilizar Ganache ya que permite tener un nodo local con una sencilla instalación y se pueden cargar contratos inteligentes desde el navegador gracias a Remix.

### *5.3.1. Ganache*

Ganache se define como una aplicación "*one click blockchain*", la cual te permite configurar una red local de Ethereum para ejecutar pruebas, comandos, inspeccionar el estado de la red, los bloques minados y las transacciones, todo desde tu ordenador.

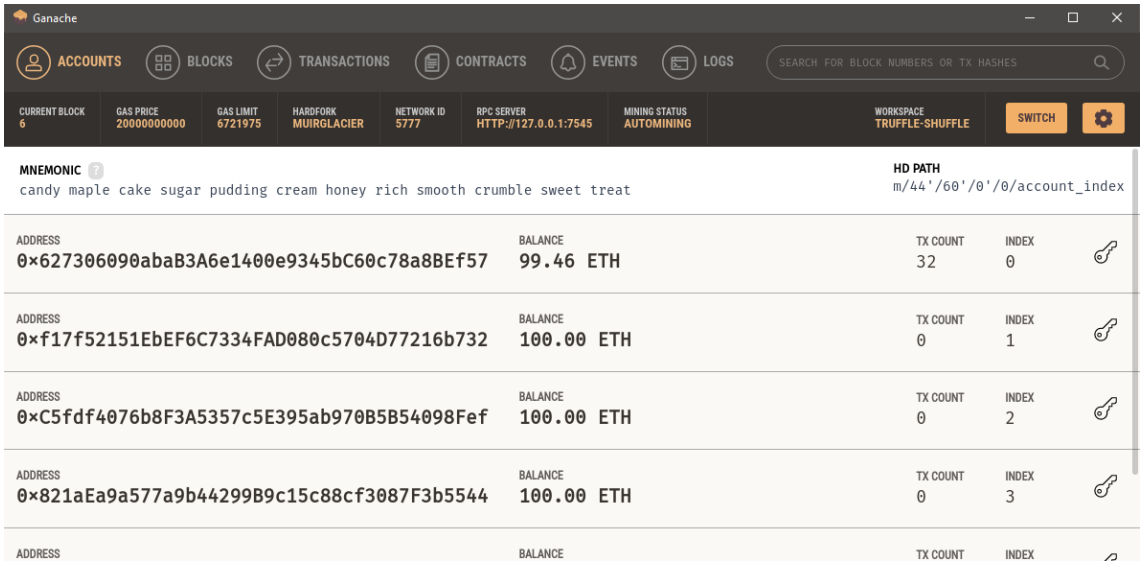


*Figura 13 - Interfaz de ejemplo de Ganache. Fuente: https://www.trufflesuite.com/ganache*

Gracias a Ganache se pudo configurar una red local que me permitiría ver todas las interacciones con la red y el funcionamiento interno de la misma. Esto sirvió más para la parte de aprendizaje sobre la red ya que para realizar la aplicación se utilizaría más adelante una red pública de Ethereum. A continuación, se explicará el contrato básico que fue creado y como se probaron sus llamadas en Ganache.

### *5.3.2. Smart Contract - Document*

Para realizar las pruebas en la red de Ganache se creó un contrato llamado *Document*. Este contrato simplemente almacenaría dos cadenas de caracteres: un nombre del documento y un autor. Estas se podrían asignar llamando a una función y se podrían devolver llamando a otras funciones.

```
pragma solidity ^0.4.24;

contract Document {

    string private documentName;
    string private authorName;


    function store(string docName, string authName) public {
        documentName = docName;
        authorName = authName;
    }

    function getDocName() public view returns (string){
        return documentName;
    }


    function getAuthorName() public view returns (string){
        return authorName;
    }
}
```

*Figura 14 - Mi primer contrato inteligente para pruebas en la red de Ganache.*

Para comprobar el funcionamiento del contrato (figura 14) primero debía ser compilado en el IDE sin ningún error. Una vez hecho, se podría conectar el IDE con un proveedor de web3 local, en este caso Ganache, y lanzar el contrato en la red (figura 15).

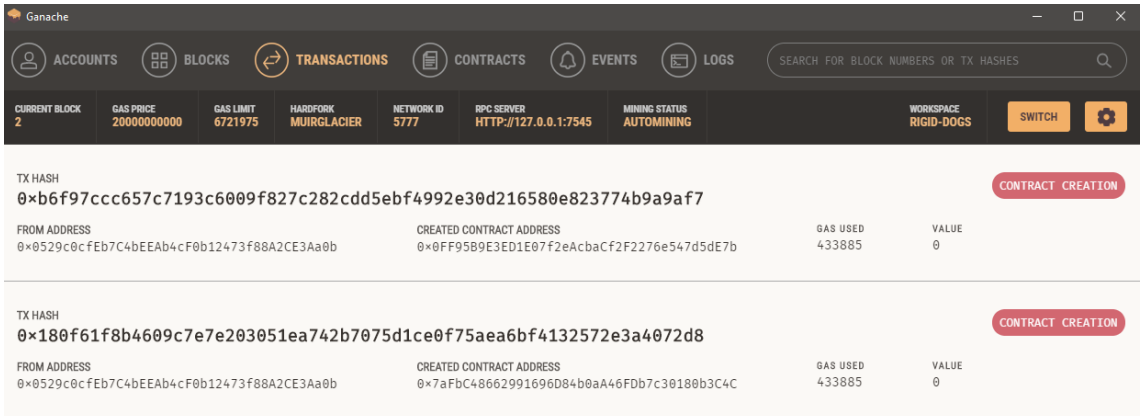


*Figura 15 - Contratos lanzados en la red de Ganache.*

Ya con el contrato funcional, desde el IDE Remix se pueden hacer llamadas a las funciones. En este caso, la función *store*, que requiere introducir dos cadenas de caracteres correspondientes al nombre del documento y al nombre del autor.

Cuando se llama a esa función desde el editor, se crea una transacción nueva. A las variables del contrato se le asignan nuevos valores y eso queda registrado en la transacción. Como se puede comprobar en la figura 16, la transacción queda registrada como *CONTRACT CALL*. Una vez registrada, en el editor se pude ver el estado de las variables. El nombre asignado a la variable de autor era "*World*" y al llamar a la función para obtener el dato se muestra que se ha asignado correctamente.

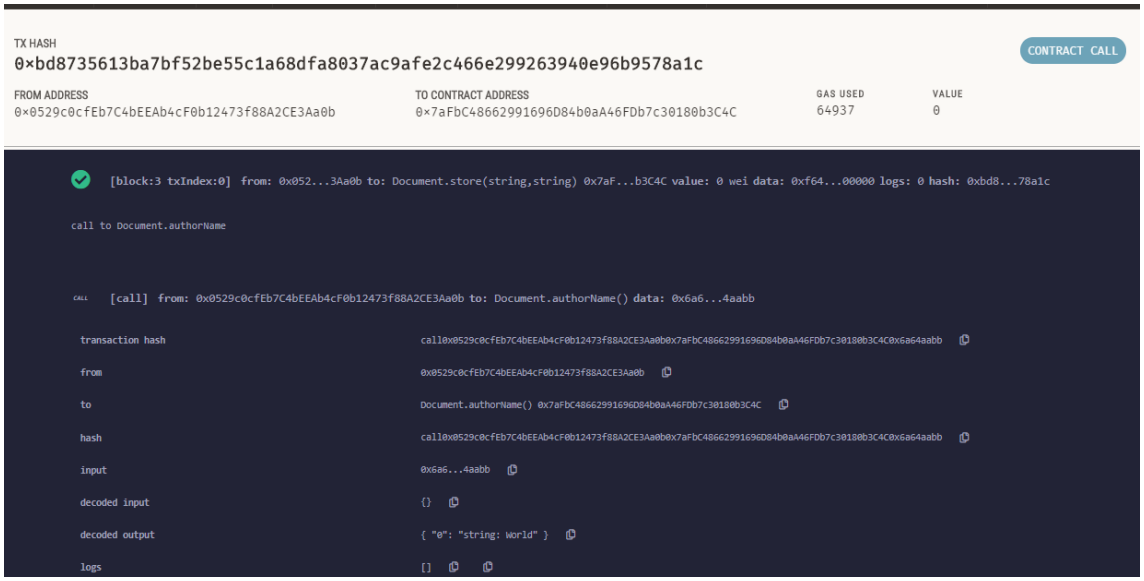


*Figura 16 - Llamada de asignación al contrato inteligente y comprobación del estado de la variable documentName.*

Con esta prueba, ya se podía comenzar a realizar nuevas pruebas en redes públicas. Para ello, se utilizaría la red de Rinkeby, una de las redes públicas más utilizadas para *testing* de aplicaciones basadas en contratos inteligentes, ya que permite visualizar todas las transacciones con un explorador de bloques online y se puede interactuar con ella directamente desde Remix.

## 5.4. *Testing* en redes publicas

Gracias a los conocimientos adquiridos sobre la red de Ethereum con las pruebas locales, se pudo pasar a realizar pruebas en las redes públicas.

Realmente, el cambio no es muy grande respecto a las redes locales, solo que para utilizar una red pública se debe disponer de una cartera con fondos en esa red. En este caso, ya que se pensó en utilizar Rinkeby, fue necesario descargar una cartera digital a la que poder añadir fondos.

### *5.4.1. Metamask*

Aquí es donde entra en juego Metamask. Este cliente de carteras digitales puede ser instalado en el navegador como una extensión y te permite tener múltiples carteras con distintas criptodivisas basadas en los tokens de Ethereum. La parte importante de este cliente es que permite conectar la cartera a la aplicación que se esté utilizando, de manera segura, ya que no se requiere de la clave privada de la cartera para interactuar con la red. Todas las transacciones que vayan a ser validadas pasan por un *pop-up* que el usuario debe aceptar, relegando así la decisión de la interacción con los contratos inteligentes en manos del usuario final.

Esta cartera sirve como puerta de entrada para utilizar las aplicaciones basadas en el blockchain de Ethereum y es la que se ha decidido utilizar en el proyecto. Facilita enormemente interactuar con la red y resulta sencillo añadir fondos de prueba para las redes como Rinkeby.

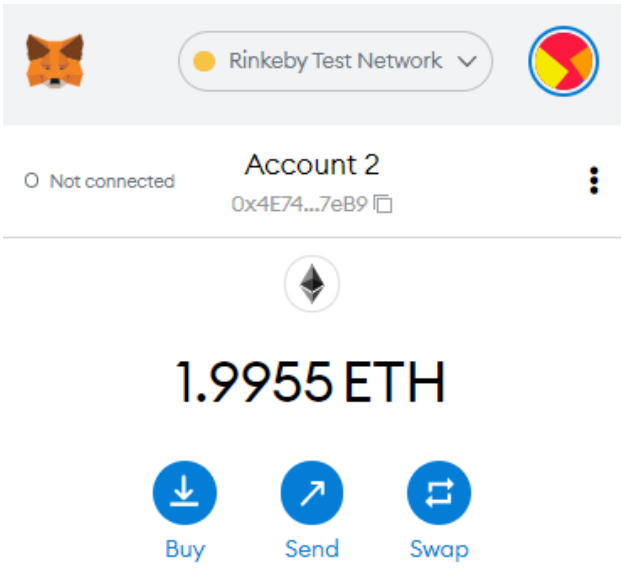


*Figura 17 - Cartera de Metamask con fondos disponibles en la red de Rinkeby.*

### *5.4.2. Faucet de Rinkeby*

Para añadir dichos fondos se necesita acceder a una de las webs de *faucet*, traducido como grifo. Estas webs se pueden definir como "un sistema de recompensas, en forma de web o aplicación, desde el cual pueden obtenerse pequeñas cantidades de una cierta criptomoneda." [9]

Normalmente, para poder optar a estas recompensas se pide una acción por parte del usuario, como, por ejemplo, poner un comentario en una de sus redes sociales con la dirección de la cartera y la página web que te los suministra.

En este caso, ya que se utilizaba la red de Rinkeby, se necesitaba un *faucet* que distribuyese fondos para esa red, concretamente Rinkeby Authenticated Faucet [10]. Simplemente, poniendo una publicación en Facebook o Twitter se pueden reclamar fondos para la cartera. Sin embargo, esto tiene un límite por horas, ya que un abuso de llamadas al servicio podría causar problemas a la red. Solo se pueden recibir las divisas en periodos de 8, 24 o 72 horas.

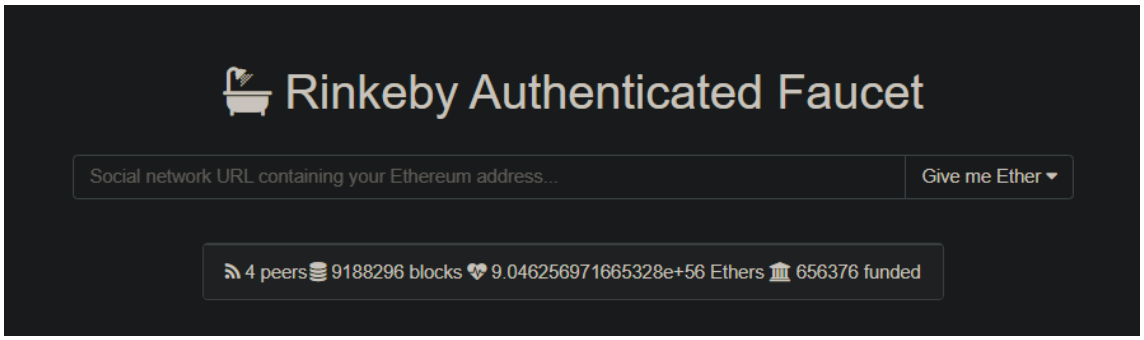


*Figura 18 - Interfaz de Rinkeby Authenticated Faucet.*

Teniendo ya los fondos disponibles en la cartera, se podría comenzar a lanzar peticiones a la red de pruebas y subir el contrato inteligente.

### *5.4.3. Pruebas en la red*

Volviendo al editor Remix, con una cuenta de Metamask y una cartera con fondos ya se podía lanzar el contrato en la red de Rinkeby e interaccionar con él.

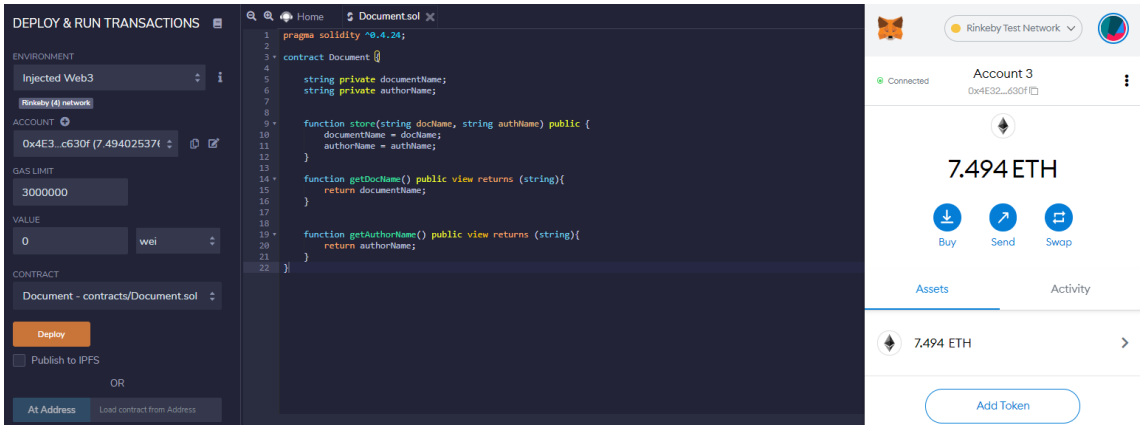


*Figura 19 - Cartera funcional en la red de Rinkeby usando Remix.*

En la figura 19 se puede ver en la parte de la izquierda que a través de una aplicación web3 como es Metamask se puede conectar con Remix y acceder a la red de Rinkeby desde el navegador. Ahí, el IDE carga la cartera con los Ethers disponibles que fueron adquiridos con *faucet* y permitió lanzar el contrato.

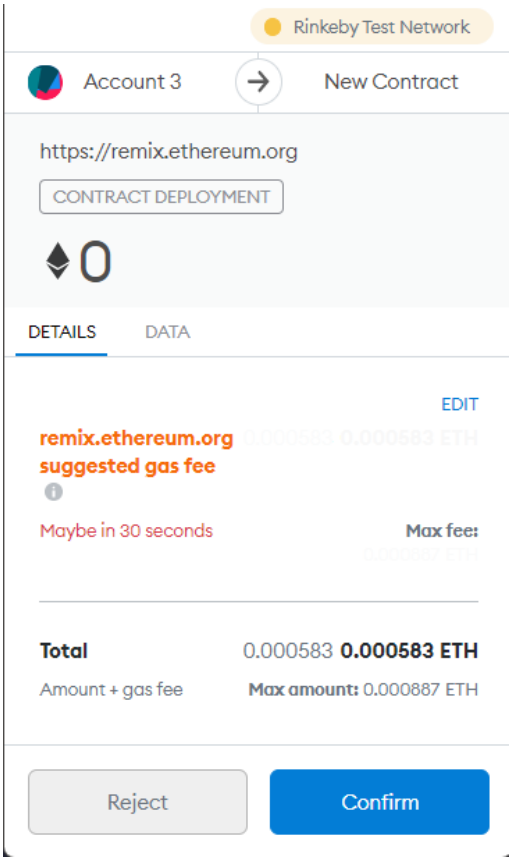


*Figura 20 - Transacción para el lanzamiento de un contrato en la red de Rinkeby.*

Para hacerlo, se requiere una pequeña cantidad de Ethers en tu cuenta, por lo que es necesario disponer de unos cuantos fondos. Una vez se pida lanzar el contrato, aparecerá un *pop-up* de Metamask indicando cual va a ser el precio en Ethers que cuesta tener funcionando el contrato (figura 20). Según la tasa de gas fijada, el tiempo en realizarse esa transacción será mayor o menor.

Con el contrato ya lanzado y aceptado por la red, se puede buscar su dirección en la red de Rinkeby para ver todas las transacciones que serán realizadas en las pruebas (figura 21).

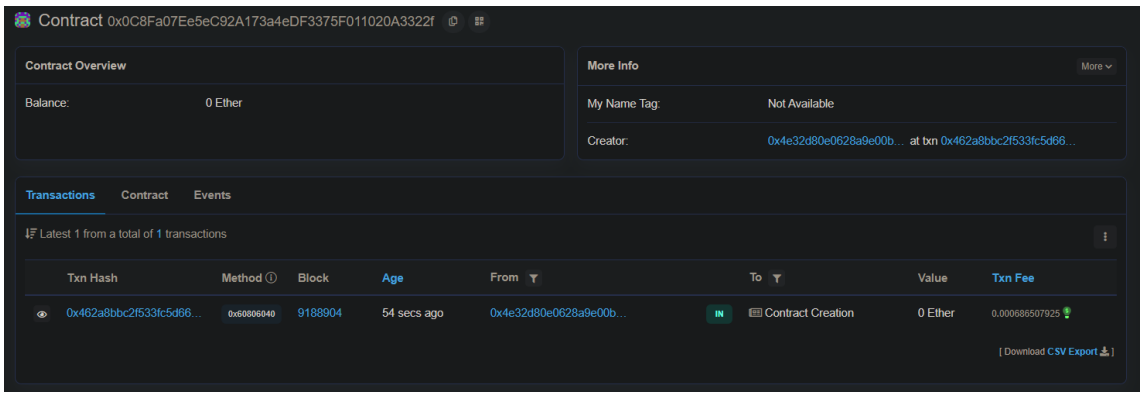


*Figura 21 - Contrato ya funcional en la red de pruebas de Rinkeby.*

Con ello, ya se podía interactuar con el contrato. Las mismas pruebas realizadas en la red local de Ethereum de Ganache deberían poder ser realizadas igualmente en la red de pruebas. La única diferencia sería el validar las transacciones a través del cliente de Metamask.

Como se puede observar en la figura 22, se consiguió almacenar "El Quijote" y a su autor, Cervantes, en una transacción de la red. Por supuesto, el contrato no aceptaba documentos por lo que los únicos datos que había eran cadenas de caracteres, pero con estas pruebas realizadas se permitió seguir avanzando en el proyecto.


*Figura 22 - Registro de una transacción con el contrato inteligente Document.*

## 5.5. Cliente Dummy

Después de las pruebas, se encomendó crear un cliente sencillo que permitiese interactuar con la red desde el navegador. La primera idea fue crear una extensión de navegador que permitiese realizar la subida del documento a la red junto a ciertos metadatos, pero debido a los requisitos del proyecto, era probable que la extensión acabase dando más problemas que soluciones, por lo tanto, fue una idea descartada.

Al final, la mejor opción sería crear una página web que permitiese hacer las subidas y las interacciones con la red sin necesidad de depender de la plataforma web en la que fuese utilizada, ya que muchas veces las extensiones web pueden tener problemas de compatibilidad en base al navegador que se utiliza, mientras que una aplicación web permite ser utilizada por cualquier tipo de navegador.

El primer problema respecto a la subida de archivos es el propio tamaño del archivo. Como se ha comentado antes, realizar una transacción en la red de Ethereum tiene un coste. Este coste viene determinado principalmente por la cantidad de datos que están adjuntos a esa transacción. Un documento o un archivo pueden llegar a ocupar varios kilobytes, por lo que acabar subiendo un pequeño documento de 100 kilobytes puede llegar a costar cientos e igual miles de dólares.

Es aquí donde IPFS (InterPlanetary File System) aporta una solución al problema.

### *5.5.1. IPFS*

El sistema IPFS o sistema de archivos interplanetario funciona como un protocolo y una red *peer-to-peer* basada en el almacenamiento y distribución de archivos de forma descentralizada. Esta red funciona de forma similar a BitTorrent, pero en vez de que ciertos usuarios sean los que distribuyan un archivo a otros, son todos los usuarios los que tienen parte de todos los archivos de la red.

Este sistema de archivos utiliza un sistema de direcciones que permite identificar cualquier archivo dentro de esa red, es decir, en el momento que un archivo sea subido a IPFS se calculara un hash único en la red que permita su identificación para así poder buscar ese archivo a través de una *gateway* o puerta de entrada.

La idea principal del sistema es permitir a los usuarios compartir los archivos entre ellos con un sistema de nodos, en el que para obtener un archivo y servirlo a un cliente, los nodos tengan que comunicarse entre sí para obtener las trazas del archivo sea donde sea que esté alojado. Esto funciona gracias a una tabla de hash distribuida (figura 23) donde los nodos comprueban en que otros nodos se alojan las partes de los archivos para distribuírselos al cliente que pide el archivo completo.

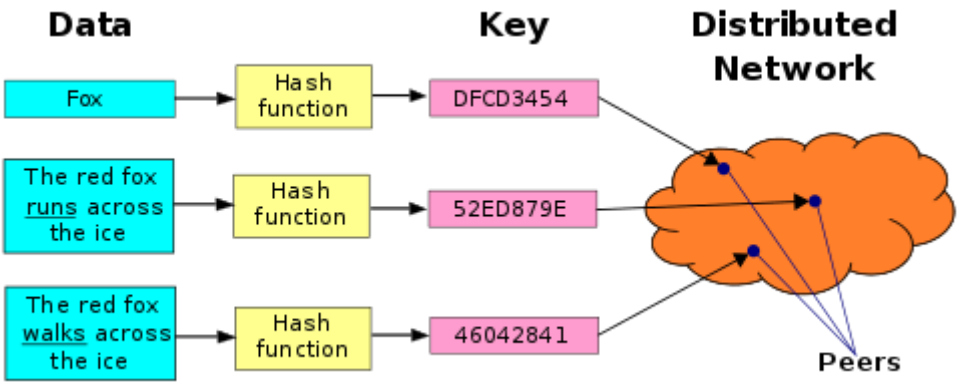


*Figura 23 - Tabla de hash distribuida. Fuente: https://es.wikipedia.org/wiki/Tabla_de_hash_distribuida*

Los archivos alojados en IPFS pueden ser obtenidos con un cliente local de IPFS que puede ser descargado en un ordenador o a través de una puerta de entrada alojada en internet. Estas puertas de entrada se pueden encontrar en el Github de IPFS [11] y cualquier archivo puede ser pedido a través de ellas, ya que solamente se necesita el hash IPFS de ese archivo para encontrarlo. Para este proyecto se utilizó la puerta de entrada de dweb, dado que funciona como una puerta segura para archivos sensibles.

Una vez entendido como funciona IPFS, es necesario usar un cliente para poder realizar subidas a la red. Ya que el objetivo era crear una aplicación web, fue necesario informarse sobre qué librerías podían ser utilizadas para ello. La más completa resultó ser JS-IPFS. Ya que se iba a trabajar con JavaScript, esta librería permitiría hacer subidas a IPFS simplemente teniendo la dirección de un nodo y que la aplicación permitiese cargar los bytes del archivo en la aplicación.

### *5.5.2. Cliente Web Básico*

Para crear el cliente web, se tuvo que buscar bastante documentación sobre cómo crear una página que permitiese interactuar con la red de Ethereum y cargar un archivo en IPFS todo desde la misma aplicación.

Gracias a cursos y documentación fue posible crear una aplicación web de Ethereum con NodeJS, React.js, Web3JS y JS-IPFS. Estas tecnologías han sido explicadas previamente en el apartado 5.1 sobre las tecnologías y herramientas utilizadas.

Con el curso, se consiguió implementar el esqueleto de una web muy sencilla que permitiese desarrollar las funcionalidades necesarias para la web final. El apartado visual, apreciable en la figura 24, es muy simple, pero a la vez efectivo para las pruebas que eran requeridas. Esta web permitía cargar un archivo para subirlo a un nodo de IPFS y poder obtener la transacción y unos cuantos datos sobre la misma.

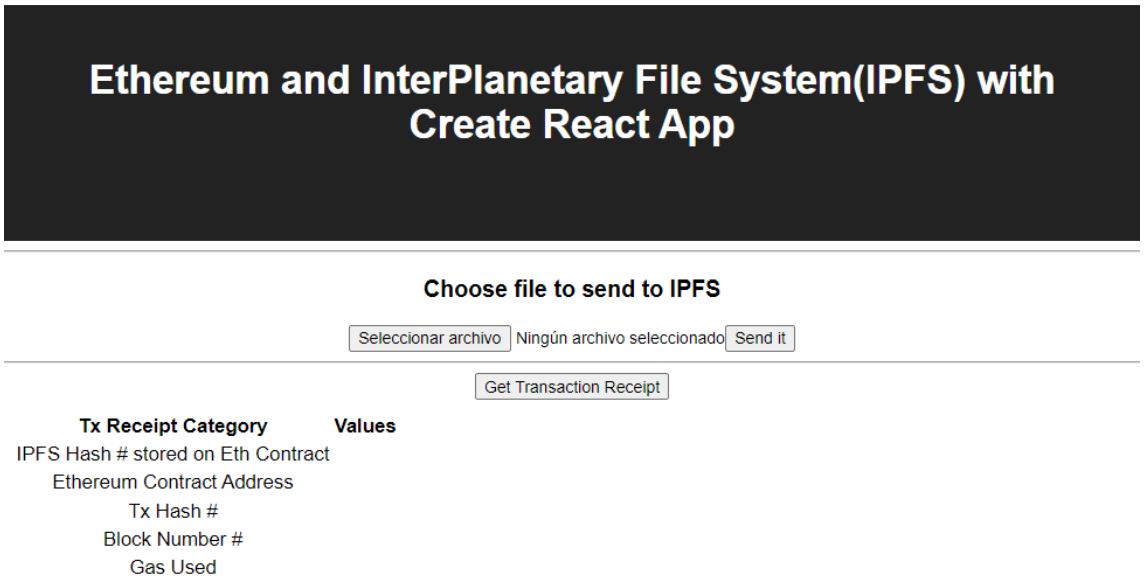


*Figura 24 - Página de pruebas para subir archivos a IPFS.*

Esta web permitió hacer pruebas de conexión con la red desde una aplicación hecha de cero, obteniendo un hash de IPFS que luego podía ser buscado en una *gateway*.

Teniendo esa parte funcional, ya que se iba a trabajar en la red de Rinkeby, era necesario un nodo de IPFS que estuviese conectado a esa red. Existen proveedores de nodos como Infura que te permiten servir como puerta de entrada para una red de blockchain de pruebas y su red IPFS.

### *5.5.3. Infura*

Infura permite crear un proyecto (figura 25) que proporciona un nodo básico al que se le pueden hacer hasta 100.000 peticiones al día. Este número de peticiones se puede ampliar con distintos planes de pago, desde 200.000 peticiones al día por $50 al mes hasta 5.000.000 de peticiones al día por $1000 al mes.

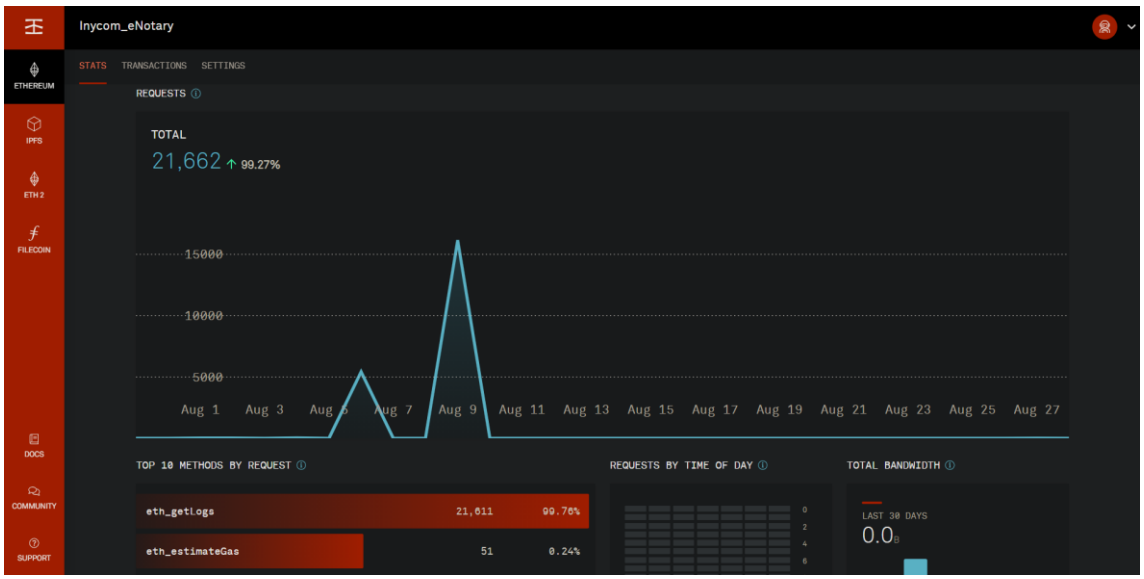


*Figura 25 - Página principal de mi proyecto en Infura.*

En este caso, ya que la aplicación no iba a necesitar más de 100.000 peticiones por día no ha sido necesario ampliar el plan por lo que no será comentado en el apartado económico.

Ya con el proyecto creado en Infura, se necesitaba el *endpoint* (figura 26) de la red que permitía conectar al nodo el cual se puede encontrar en la pestaña de *Settings* del proyecto. La dirección de este *endpoint* tiene que ser introducida en el código al crear una nueva instancia de web3. Cuando se crea una nueva instancia, el constructor de la clase requiere que se

introduzca un proveedor de la red y en este caso se utiliza el *endpoint* de Infura como proveedor (figura 27).


*Figura 26 - Endpoints de mi proyecto en Infura.*

```
const infuraEndpoint = "https://rinkeby.infura.io/v3/
87d33b235b06421584ca2ce2d26ea056";
const web3 = new Web3(infuraEndpoint);
//My Infura project rinkeby endpoint.
```

*Figura 27 - Endpoint HTTPS de mi nodo utilizado para crear una instancia de Web3JS.*

Una vez configurado el nodo y cargado el *endpoint* en la aplicación, todas las peticiones que se realizasen a la red de Rinkeby pasarían por él, lo cual permite analizar y comprobar que tipo de peticiones se están mandando.

Ya con la conexión a la red funcional, el siguiente objetivo era crear un *Smart Contract* que realizase las acciones necesarias para que funcionase como una notaría digital.

### *5.5.4. Smart Contract - StoreDocument*

Durante esta parte del desarrollo se debatió sobre la estructura final que debía tener el contrato inteligente y cómo debía ser implementado.

Antes de comenzar a hacer pruebas con los eventos, los esfuerzos se centraron en desarrollar un contrato inteligente más complejo que integrase la lógica de suministrar los documentos a los usuarios dentro de él. Desarrollar así un contrato tiene su coste, ya que conforme más

variables se almacenan en el contrato, más acaban siendo modificadas. Al final, actualizar variables supone un coste mayor para el usuario final ya que la transacción requiere más gas.

El contrato que fue desarrollado se centraba en la gestión de los documentos con unas cuentas de administrador llamadas *manager* que fuesen las únicas que pudiesen obtener el hash que subía a la red en esa transacción.

```
address                         public  contractOwner;
mapping(address=>address)       public  manager;
address                         public  txOwner;
string                          private _hash;

constructor() public{
    if(contractOwner == 0x0){
        contractOwner = msg.sender;
        txOwner = msg.sender;
    }

    else
        txOwner = msg.sender;

}
```

*Figura 28 - Variables y constructor de StoreDocument.*

En la figura 28 se pueden observar las 4 variables que había y el constructor. Para empezar, habría un *contractOwner* el cual sería la dirección de la cartera desde la que subía el contrato. Solamente esa dirección podía añadir nuevas direcciones como *managers* por lo que ningún otro usuario final podría comprobar datos de otros usuarios. Aquel que realizase la llamada a una función que asignase el hash se registraría como *txOwner* para luego poder ver loo que ha subido a la red.

En el constructor se asignaba el *contractOwner* al primero que cargase el contrato en la red y para las siguientes llamadas a las funciones solamente se asignaría el *txOwner*. Una de las ventajas de Solidity es que cuando se manda una petición al contrato inteligente podemos obtener ciertos parámetros de la petición desde la variable *msg*. Esta dispone de propiedades como *sender* que almacena la dirección de la cartera que manda esa petición o *data* que almacena todos los datos de la petición.

```
//Setters
function addManager(address _add) ownership public {
    manager[_add] = _add;
}

function registerHash(string newHash) public {
    _hash = newHash;
}

//Getters//

function getHash() manage public view returns (string _hash){
    return _hash;
}

// Modifiers
modifier ownership(){
    require(msg.sender == contractOwner);
    _;
}

modifier manage(){
    require(msg.sender == manager[msg.sender] || msg.sender == contractOwner || msg.sender == txOwner);
    _;
}
```

*Figura 29 - Funciones y modificadores de StoreDocument.*

Las funciones del contrato (figura 29) permitían añadir un nuevo *manager* a la lista, registrar un hash u obtener el hash de esa transacción. Lo importante de estas funciones son los modificadores que tienen y la lógica que añaden.

Los modificadores en Solidity son un tipo de declaración de las funciones con los que se le añade una comprobación extra antes de ser ejecutadas. Por ejemplo, en este caso se dispone de dos modificadores: *ownership* y *manage*. En el caso del primero, requiere que, si una función tiene ese modificador, aquel que la llama debe tener la misma dirección de cartera que el poseedor del contrato, o de otra forma, la función no será ejecutada. Lo mismo ocurre con el segundo, salvo que *manage* está creado para comprobar quienes pueden tener acceso a los hashes de los documentos subidos.

Previamente se habían hecho pequeñas iteraciones sobre este mismo contrato, al principio sin *managers* y sin modificadores, únicamente disponiendo de funciones que asignasen el hash del contrato a la variable, pero conforme se fue discutiendo que cualidades debía tener el contrato en las reuniones, la implementación fue cambiando.

Con el nuevo contrato funcional en Remix, se necesitó añadir el contrato al código de la web con el ABI. Para hacer llamadas a las funciones del contrato se requiere de un ABI o Interfaz Binaria de Aplicación (figura 30) la cual replica las especificaciones del contrato tales como las variables de las que dispone, el tipo de funciones, lo que modifican, lo que devuelve, etc. Este, junto a su dirección en la red, permite hacer una instancia de un *Contract* de Web3 al que se les pasarán las llamadas a sus funciones.

```
const address = "0x07FBac5868221462A45A0D1a7905213Ca086A5bd";
const abi = [
    {
        "constant": false,
        "inputs": [
            {
                "name": "rHash",
                "type": "string"
            }
        ],
        "name": "registerHash",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "name": "owner",
                "type": "address"
            },
            {
                "indexed": false,
                "name": "_hash",
                "type": "string"
            }
        ],
        "name": "RegisterHash",
        "type": "event"
    }
];

export default new web3.eth.Contract(abi, address);
```

*Figura 30 – Ejemplo de ABI del contrato que he utilizado para la página web final.*

Al hacer pruebas para subir el hash IPFS de un documento a la red salió un problema sobre cómo realizar las llamadas. Con Web3JS las llamadas pueden ser realizadas con *call* o con *send*. Estos dos se diferencian en la forma que tienen de ejecutar la función. Con *call* se hace una llamada a la función constante dentro de la máquina virtual EVM sin mandar ninguna

transacción a la red, lo que permite realizar pruebas internas u obtener el valor de una variable con un *getter*. Sin embargo, cuando se realiza una llamada con *send*, esta sí que manda una transacción al contrato inteligente y permite alterar el estado del contrato, cosa que era necesaria para el contrato desarrollado. El error viene al pasar por el nodo de Infura, ya que debido a las especificaciones de su API no se permite realizar llamadas con *send* a sus nodos. Estas llamadas solamente pueden ser realizadas con una transacción en su estado *raw* o crudo (figura 31), las cuales requieren de parámetros similares a los que aparecen en una transacción como se ha explicado en el apartado 5.2.1.

```
const myData = storedocument.methods.registerHash(hash).encodeABI();
const txParameters = {
    nonce: '0x00', // ignored by MetaMask
    gasPrice: '0x3B9ACA00', // customizable by user during MetaMask confirmation.
    gas: '0x' + estGas, // Gas Calculator
    to: ethAddress, // Required except during contract publications.
    from: ethereum.selectedAddress, // must match user's active address.
    value: '0x00', // Only required to send ether
    data: myData, //Smart Contract Method
    chainId: '0x3' //Ignored by Metamask
};
```

*Figura 31 - Transacción cruda con los datos requeridos para ser lanzada en la red.*

Con este tipo de transacción, el nodo de Infura permite realizar una llamada utilizando *ethereum.request,* el cual recibe como parámetros de entrada el método que queremos llamar y los datos de la transacción. Para ello, se debe usar el método *'eth_sendTransaction'.*

### *5.5.5. ¿Stateless o Stateful?*

En este punto se disponía de un contrato inteligente que almacenaba unas variables que cambiaban a cada transacción realizada. Esto podía suponer un problema ya que la trazabilidad de los valores de estas variables se volvía más complicada. Para poder obtener los datos que se enviaban a la transacción dentro de la red, esta debía disponer del contrato inteligente de una forma pública, algo que por una parte puede hacerlo más inseguro ya que cualquier usuario podría ver el estado de todas las transacciones y los valores que le han sido asignados, y a su vez cualquiera podría copiar el código del contrato y hacerlo suyo.

Todo esto supone un problema a la hora de desarrollar un contrato inteligente y para los requisitos que tenía el trabajo se podía solventar de diversas maneras.

La primera, que se siguiese con un contrato *stateful,* es decir, que almacenase las variables en el contrato y continuar con el requisito de que la red disponga del código fuente del contrato inteligente. Para ello se debía publicar el contrato dentro de la red en Rinkeby, de modo que se pudiese desencriptar el código binario del input mandado.

La segunda, crear un contrato *stateless* que no almacenase ningún estado y solamente recibiese llamadas. Esto puede ser realizado creando un contrato que solo tenga funciones, las cuales puedan ser llamadas desde el código, y que, al crearse la transacción, se queden reflejados los datos que han sido introducidos, aunque no se asigne nada a ninguna variable. Otra de las formas, la cual es la más llamativa, es a través de eventos de Solidity. Los eventos ya han sido previamente explicados en el apartado 5.2.1 pero ahora se va a tratar más el aspecto de código que los concierne.

Para que un *Smart Contract* pueda lanzar un evento necesita de la declaración de dicho evento con su nombre y las variables que requiere de entrada. Luego, sería necesario crear una función que haga de *emitter*. Esta función debe tener dentro una palaba reservada '*emit*' con el nombre del evento al que va a llamar con los valores de entrada (figura 32)

```
event RegisterHash(address indexed owner, string _hash );

//Event Emitter
function registerHash(string rHash) public {
    emit RegisterHash(msg.sender, rHash);
}
```

*Figura 32 - Evento en Solidity.*

En este caso, se eligió simplificar al máximo la funcionalidad del contrato para que utilizase la lógica más simple dentro del blockchain y así poder evitar gastos mayores a la hora de subir documentos a la red. Respecto a los costes que supone tener un contrato *stateful* y un contrato *stateless* se hablará en el apartado del estudio económico.

### *5.5.6. Manipulación de los eventos.*

Una vez terminado el contrato que iba a ser utilizado definitivamente, el siguiente paso era realizar el código necesario para la implementación final de la subida de los archivos. La idea principal fue que en la transacción se mandase un único hash que estuviese encriptado sobre aquello que interesase guardar sobre el documento.

En este caso, se decidió que ese hash sería la suma del nombre que se quiere que tenga el documento en la red, el autor del documento, la hora a la que ha sido emitido, y el hash IPFS del documento, todo en una misma cadena de caracteres, la cual luego sería encriptada con una clave privada que solo está disponible en el código fuente de la aplicación.

La transacción final de la aplicación tendrá los *logs* de eventos en los que se encontrará el hash encriptado que hemos calculado en la aplicación, haciendo así una única subida a la red y ninguna modificación de variables.

Luego, para poder obtener los datos de estos eventos, fue necesario informarse sobre cómo se podían manipular estos datos y qué maneras había para recibirlos en el código.

Una de las opciones es suscribir la aplicación al evento que se emite. Esto significa que mientras la aplicación esté funcionando, cada vez que se emita un evento, este se quedaría guardado dentro de la aplicación, y una vez obtenido, podría ser subido a alguna base de datos que almacenase los eventos de las transacciones.

Esto por supuesto, añade un coste extra ya que habría que tener una base de datos funcional, la cual puede ser atacada, y en el momento que la aplicación requiera de una transacción, esta deberá hacer una petición a dicha base de datos, lo cual no se puede saber cuánto tiempo puede tardar ni hasta qué punto puede ser eficiente almacenando datos de las transacciones.

La otra opción es pedir todos los eventos directamente a la red de blockchain en la que esté alojado el contrato inteligente, lo cual es gestionado por la red y evita el riesgo de que los datos hayan sido manipulados e invertir más costes en alojar una base de datos. Al pedir los eventos registrados en el contrato se pueden aplicar filtros que faciliten la búsqueda gracias a que una variable del evento sea indexada. En este caso, se decidió hacer la variable de *address*, que almacena la dirección del emisor del documento, como variable indexada, la cual permitiría realizar búsquedas que devolviesen todos los eventos que haya realizado el usuario con dicha dirección.

El gran inconveniente que tiene este método es la llamada que realiza, ya que a más grande se vuelvan los registros de los eventos en el contrato, más tiempo tardará en encontrarlos. Al final, es una petición que se le realiza a la cadena de bloques y aunque sea una simple petición que no asigna ningún dato puede ralentizarse en un futuro, debido a la incertidumbre que supone el

futuro de la red de Ethereum y su funcionamiento en unos años. Además, según la cantidad de llamadas realizadas al día por la aplicación puede llegar a aumentar el coste requerido por el nodo, ya que se debería ampliar la cantidad de llamadas por día disponibles, aunque este caso es muy extremo debido a que es complicado que se realicen más de 100.000 llamadas a la red para obtener los eventos en el transcurso de un día.

Respecto al límite de llamadas, sí que es verdad que puede suponer un problema de seguridad ya que, si un usuario mandase un ataque de denegación de servicio al nodo de Infura, podría llegar a alcanzar el número máximo disponible de llamadas en un mismo día y dejar al resto de usuarios sin capacidad de registrar nuevos documentos ni comprobar los que tienen subidos en la red. Por supuesto, la seguridad del proyecto es importante, pero se requiere un gran esfuerzo para evitar este tipo de problemas los cuales serán tratados como una parte de trabajo futuro para mejorar la aplicación.

Finalmente, se eligió utilizar el segundo método ya que realizar esa petición a la red no supone ningún problema en el estado actual del proyecto y aporta rapidez y seguridad a la forma en la que se obtienen los registros de eventos.

En el código, se hace una petición con *getAllEvents()* a la que se le puedeaplicar un filtro. En este caso, el filtro que se utiliza es la dirección de cartera del usuario que hace la petición. Esta devuelve una colección de eventos de dicho usuario la cual luego es organizada en un array y se desencripta la información utilizando la clave privada. Existe una excepción al caso y es que hay carteras introducidas en el código que pueden funcionar como auditores y pueden obtener todos los documentos que han sido enviados al contrato inteligente.

Teniendo la disponibilidad para subir documentos y recibirlos en el cliente, se propuso un generador de carteras dentro de la web que permitiese a los usuarios tener su propia cartera con una clave privada distribuida por el programa. Esto sería la última parte de la implementación del cliente *dummy.*

### *5.5.7. Generador de wallets.*

La última característica de la aplicación era la creación de carteras a través de la web. Por supuesto, el cliente web no almacenaría ninguna clave privada. Es conocido que algunos generadores de carteras online han sufrido ataques y debido a que almacenaban las claves privadas de las carteras que generaban, los hackers han podido acceder en sus dispositivos a

esas carteras y transferir los fondos a sus cuentas, llegando a obtener hasta miles de dólares en criptomonedas. Uno de los casos más recientes es el de BitcoinPaperWallet, en el que se ha llegado a robar 124.85 BTC debido a que la web disponía de una característica en su servidor que le permitía compartir las claves privadas de los usuarios a aquellos que las pidiesen, la cual los atacantes abusaron para robar fondos de las carteras de los usuarios. [12]

Debido a los peligros que supone almacenar la clave privada de una cartera, la decisión para el proyecto fue relegar la responsabilidad del almacenamiento de dicha clave en el usuario. Cualquier usuario que perdiese esa clave no podría recuperar su cartera, por lo que es importante que todos los usuarios que creen una cartera con el generador guarden esa clave en un papel o en un documento. Esto es avisado en la página web definitiva dentro de la sección del generador.

Crear nuevas carteras supone un coste trivial para el cliente web, ya que utiliza una simple llamada con Web3JS a la que se le puede introducir una entropía en forma de cadena de caracteres para tener una mayor aleatoriedad y personalización a la hora de crear la nueva cartera. Una vez creada, se le devuelve al usuario la dirección de su cartera y la clave privada para que la introduzca en su gestor de carteras como Metamask.

Este supone el último cambio del cliente base y al recibir el visto bueno por parte de Inycom, se pasó a desarrollar la página web final que implementaría todo lo que he desarrollado.

## 5.6. Desarrollo de la aplicación web

En este apartado se comentará principalmente cómo fue el desarrollo de la nueva web y los problemas que se tuvieron durante el mismo, y las decisiones del desarrollo de la interfaz.

### *5.6.1. Creación del esqueleto de la web*

Con la base implementada, el siguiente objetivo era crear una web desde cero. La idea principal era una web que reaccionase a los cambios de tamaño, que se pudiese ajustar a pantallas de distintos dispositivos y que a su vez dispusiese de elementos útiles como una barra de navegación o una página principal estilizada con enlaces directos a las secciones de la aplicación.

Para ello se decidió seguir un curso de React.js que enseñara como diseñar una página web desde cero y que fuese adaptable a otros dispositivos, además de implementar *smooth scrolling*, una técnica que permite al usuario moverse por la página entre secciones con mayor comodidad sin cambios bruscos.

En este curso se aprendió también a implementar CSS con una librería de Node.js llamada *styled-components* que permite crear componentes utilizables por React.js estilizados, combinando ES6 y CSS, y pudiendo reutilizar estos componentes por todo el proyecto. Esta librería es utilizada por grandes compañías en sus páginas webs [14] y ha resultado realmente útil a la hora de realizar cambios visuales a la web. Es mucho más intuitiva que CSS tradicional y permite a usuarios nuevos adaptarse a una nueva forma de personalizar las webs.

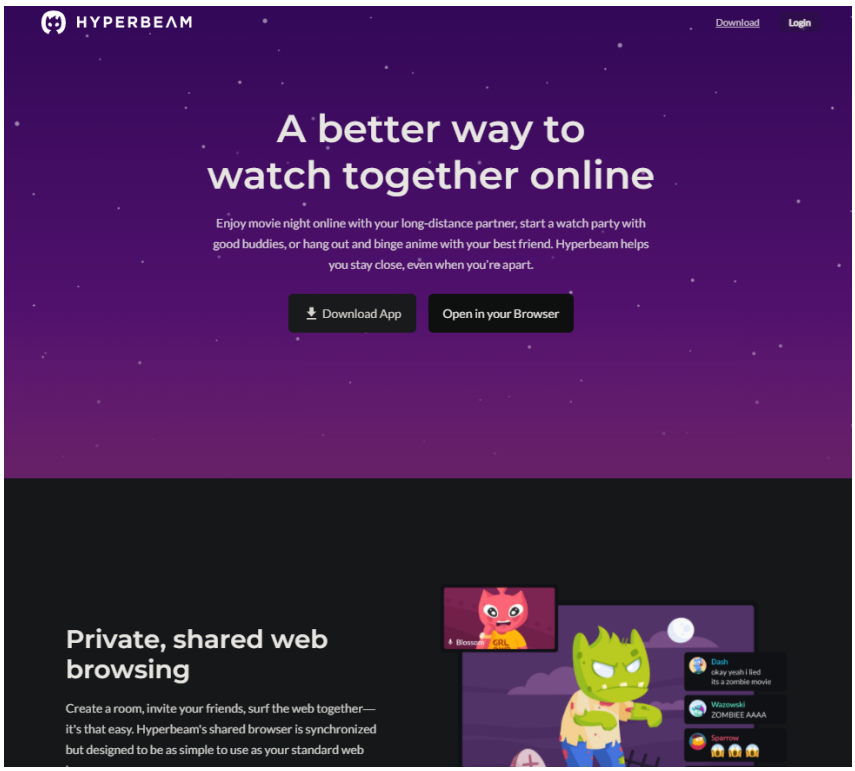


*Figura 33 - Web principal de Hyperbeam.com.*

Para el diseño se buscaron influencias en webs que utilizan un estilo limpio y de una sola página. Por ejemplo, la web de Hyperbeam.com (figura 33) implementa un diseño sencillo con unas pocas imágenes y animaciones que resultan atractivas a la vista y centran la atención del usuario en la página principal, sin muchos botones ni distracciones.

En la página principal de la web se explicarían los pasos para conectarse con Metamask, realizar subidas a la red, crear una cartera y poder visualizar los documentos que han sido subidos.

Luego cada apartado de la implementación que fue realizado en el cliente *dummy* pasaría a una página separada de la principal, manteniendo así cada característica aislada de las demás, por lo que se tendría una página para subir un documento, otra para poder revisar los documentos que han sido subidos por el cliente y finalmente una que permitiese al usuario crear su propia cartera introduciendo una cadena de caracteres para darle mayor aleatoriedad.

Ya que Metamask maneja las carteras del usuario y se requiere una cuenta para iniciar sesión en la extensión no ha sido necesario crear un registro de usuarios dentro de la página web.

Tras realizar el curso y aplicar las nuevas técnicas conocidas para el desarrollo web, la página principal se quedaría como se puede ver en la figura 34. Su nombre final sería "ethNotary" y dispone de varias secciones para tratar de enseñar al usuario las características que tiene la aplicación web. Cada sección tiene un botón que redirigirá al cliente a la página especifica de la web, exceptuando el botón de *"Install Metamask"* , que redirige directamente a la página oficial de Metamask donde puede ser descargado.

La página web dispone de una barra de navegación que se vuelve opaca cuando el usuario comienza a bajar por la página y a su vez esta se convierte en una barra lateral en caso de que el tamaño de la pantalla no sea suficiente para mostrar la barra de navegación completa. Los enlaces que tienen dichas barras dirigen al usuario a la sección donde puede encontrar una breve descripción de la característica que va a utilizar y su botón que enlaza a su página correspondiente.

Mientras el usuario navegue y le dé clic a uno de los elementos de la barra de navegación, se ejecutará un movimiento suavizado de bajada o subida en la página gracias al *smooth scrolling* que se puede implementar en React.js con *animateScroll* de la librería '*react-scroll'.*

Una vez acabado el diseño del esqueleto principal de la web, se dedicó el resto del tiempo a comenzar a implementar el resto de las páginas que tratarían la subida de archivos en la red, la obtención de estos y la creación de una cartera de Ethereum.

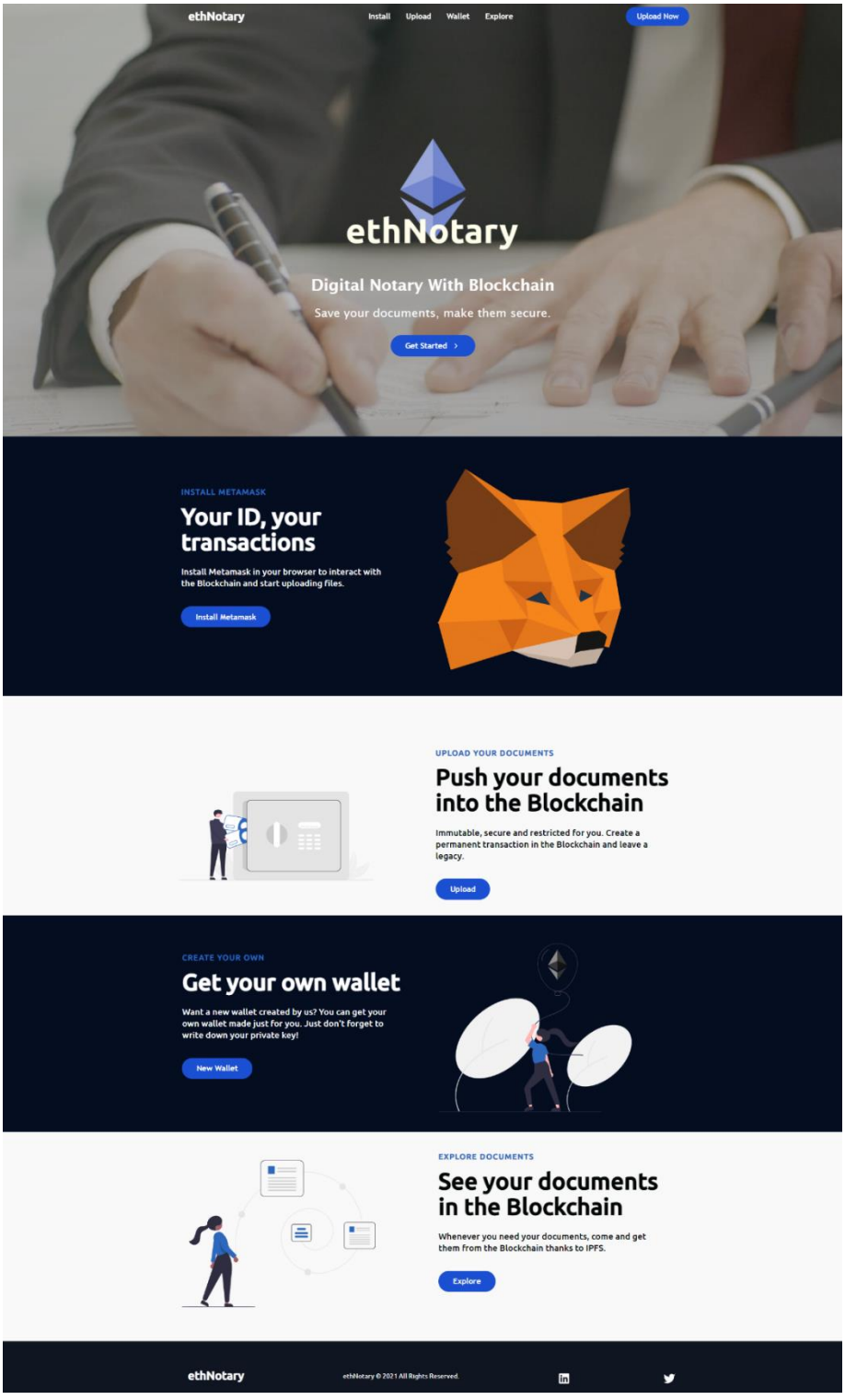


*Figura 34 - Página web principal de ethNotary.*

### *5.6.2. Implementación de las características.*

Para implementar el resto de las características de la web se tomó código del cliente *dummy*, concretamente el apartado relacionado con IPFS y Web3JS, ya que las conexiones y las funcionalidades iban a ser las mismas.

Se separó el desarrollo en tres partes: Crear primero la página que genera carteras para el usuario y le instruye como añadirlas; otra página que implementa la carga de documentos en un formulario web para poder subirlos a la red; y finalmente, un apartado en el que el usuario pueda comprobar los documentos que ha subido.

Para estas páginas se planeó que la funcionalidad y los elementos estuviesen centrados en la pantalla y que no tuviese que moverse el usuario, simplemente con un recuadro rectangular en el medio con la información necesaria para mostrar los formularios y los datos.

En el apartado del generador de carteras fue sencillo ya que simplemente se debía crear un formulario que recibiese una cadena de caracteres y cuando el usuario lo enviase se devolvería la dirección de la cartera y su clave privada junto a un pequeño infográfico sobre cómo agregarla a su cuenta de Metamask.

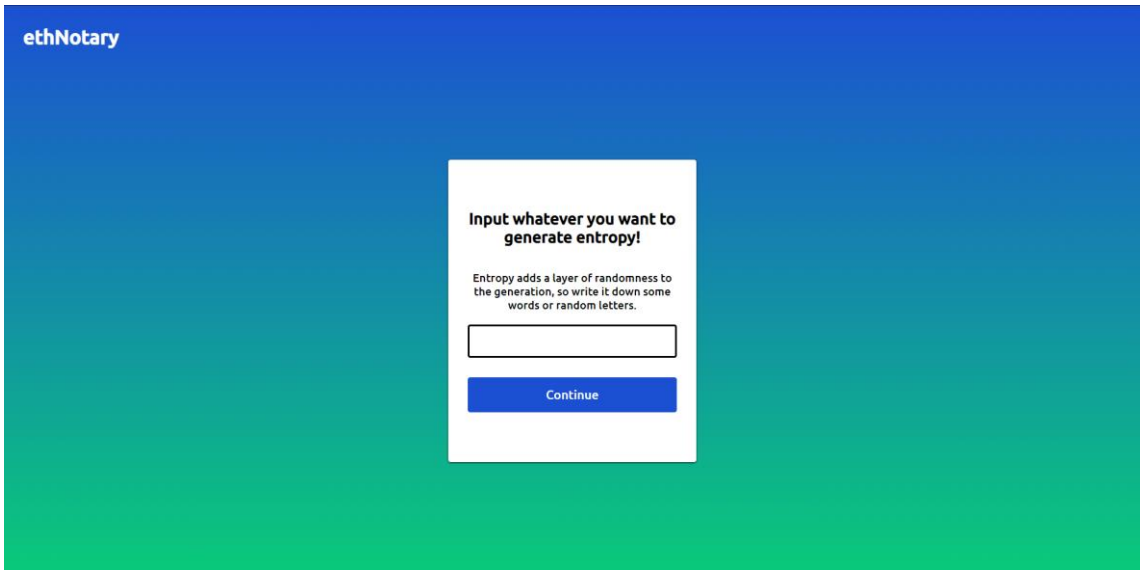


*Figura 35 - Generador de carteras: Estado 1*

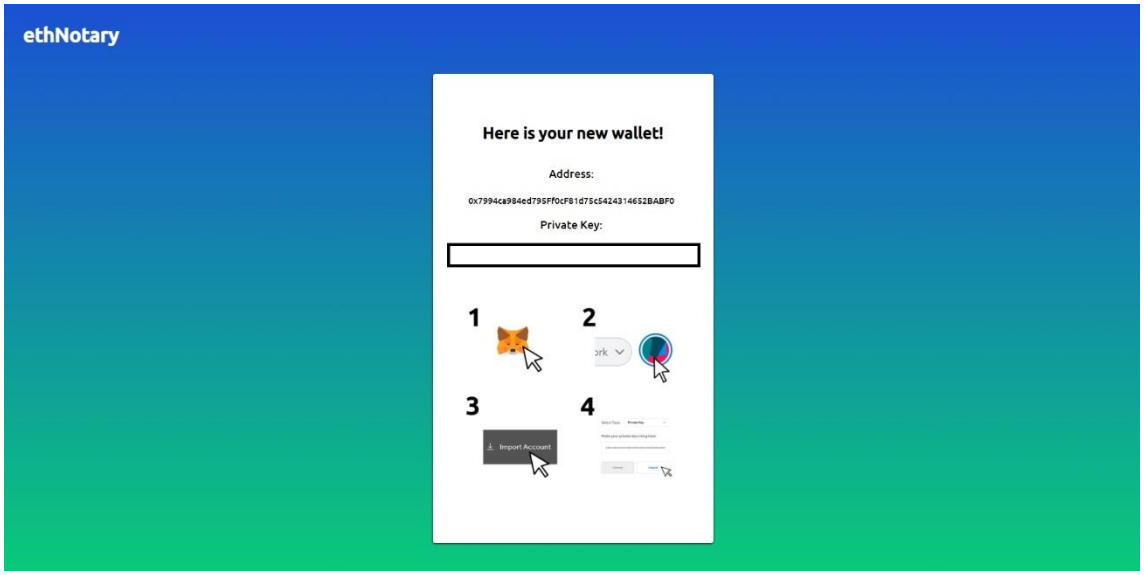


*Figura 36 - Generador de carteras: Estado 2*

En la figura 35 y 36 se pueden observar los dos estados de la página, uno para introducir el texto como entropía y el otro donde el usuario recibe las instrucciones para añadir su cartera.

En este apartado no hubo ninguna complicación salvo la implementación del segundo estado, ya que para ello se debía realizar un cambio en el estado de la página. Para ello, simplemente se añadió otro tipo de recuadro que cambiase en el momento que el usuario ejecuta la función para crear su cartera.

Para el apartado de subir el documento a la red de blockchain se complicó algo más debido a que se necesitaba crear una funcionalidad extra para la página. En caso de no tener Metamask instalado se le exigiría al usuario que volviese al primer paso y lo instalase para poder utilizar la subida de documentos. Esto fue implementado debido a que la validación de Metamask puede ser ignorada si no existe la extensión en el navegador, por lo que no tendría mucho sentido que se intentase interaccionar sin un método de conexión con la red.

En la figura 37 se encuentra el contenido que aparece como alerta para el usuario para que instale Metamask en su navegador una vez entra en la página de subida de documentos. Esto también ha sido reutilizado en el apartado de explorar los documentos subidos.

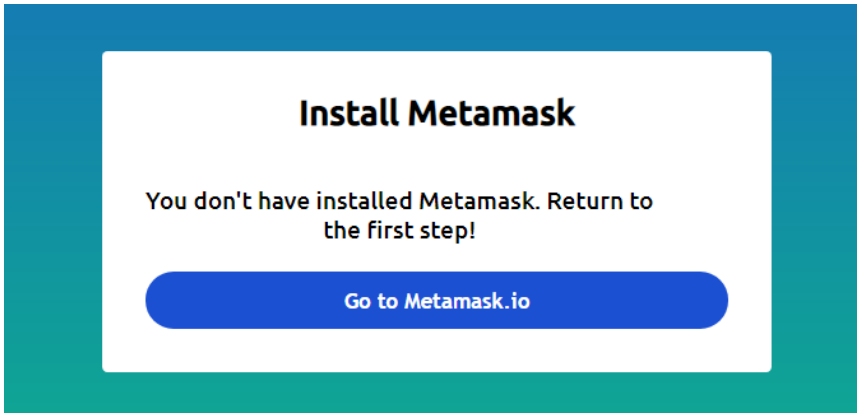


*Figura 37 - Subida de documentos: Alerta de instalación*

En caso de que el cliente tenga instalada la extensión, aparecerá un *pop-up* en el que se le pedirá que introduzca la contraseña de su cuenta de Metamask. Este es el método de seguridad del que dispone la extensión para no tener que utilizar las claves privadas dentro de las aplicaciones. Simplemente, el usuario puede desbloquear y conectar su cartera utilizando la contraseña con la que se ha registrado en la página oficial de Metamask o utilizando la clave de recuperación de su cuenta.

Una vez desbloqueada su cuenta, puede introducir los datos que van a ser necesarios para la subida del documento, en este caso, un nombre, un autor y el archivo (figura 38).

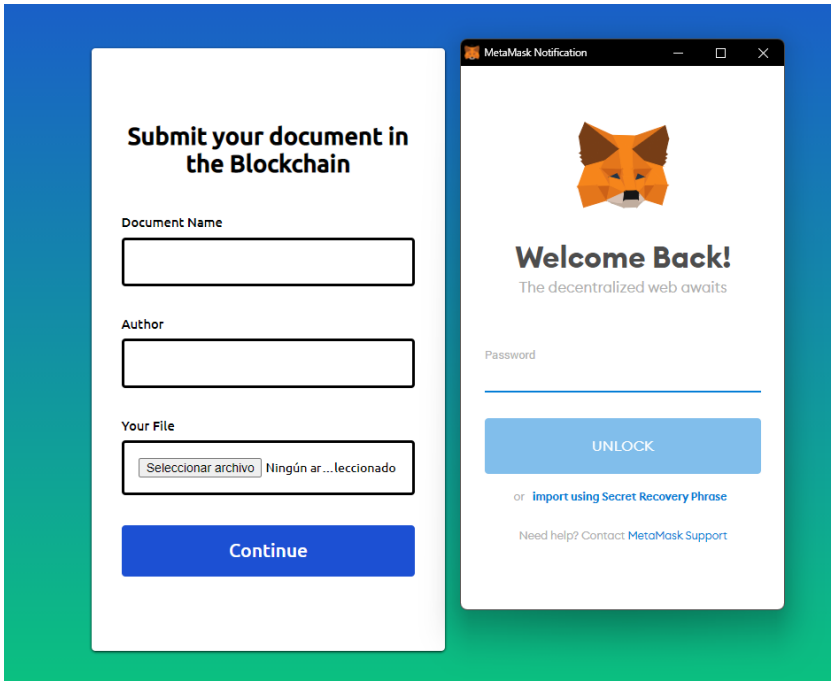


*Figura 38 – Subida de documentos: Pop-up de Metamask y formulario.*

Al ser enviado, se le pedirá al usuario que acepte la transacción en otro elemento emergente de Metamask y una vez la transacción sea aceptada se le devolverá el hash de la transacción. Esta puede tardar en ser validada debido a las tasas utilizadas como incentivo, pero estas pueden ser personalizadas por el usuario para que la transacción sea más rápida. En la figura 39 se puede observar la petición que hace Metamask con las tasas aplicadas, la respuesta de la página con el hash de la transacción, y la transacción en Etherscan, donde aún se ve que está pendiente.

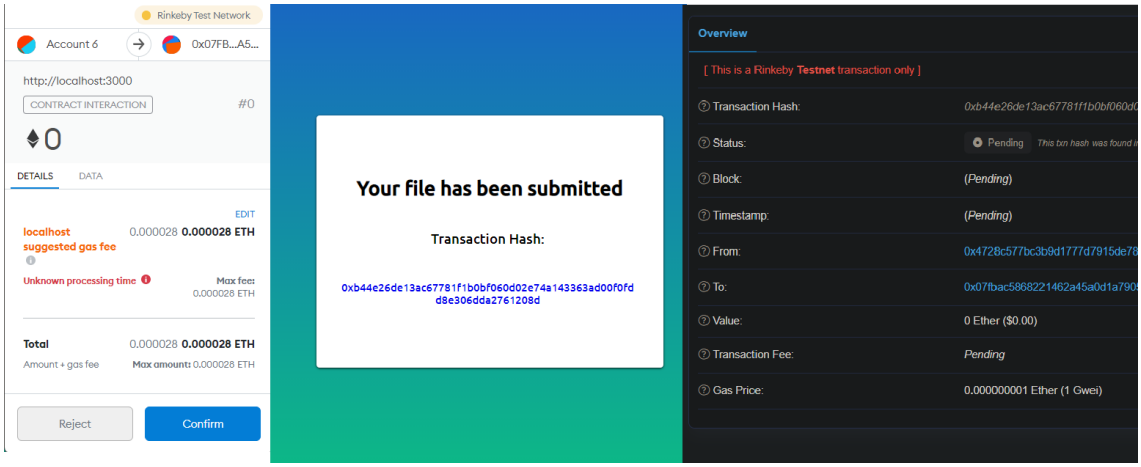


*Figura 39 - Subida de documentos: Transacción de Metamask, resultado en la web y estado de la transacción.*

Finalmente, después de la implementación de la subida de documentos solo quedaba poder devolvérselos al cliente desde la red de blockchain.

La página de exploración de documentos ha resultado más problemática que el resto ya que la idea de cómo cargarlos puede resultar en diversos errores.

La idea principal era que los documentos del usuario fuesen cargados directamente tal cual entrase a la página, pero debido a que se requiere de la contraseña de Metamask para conectar con ella podría dar cabida a algún problema en la interfaz. Otra de las ideas era implementar una pantalla de carga hasta que el usuario validase la conexión. Al final se decidió pedirle al usuario que hiciese clic en un botón una vez estuviese conectado para cargar los documentos en la página (figura 40).

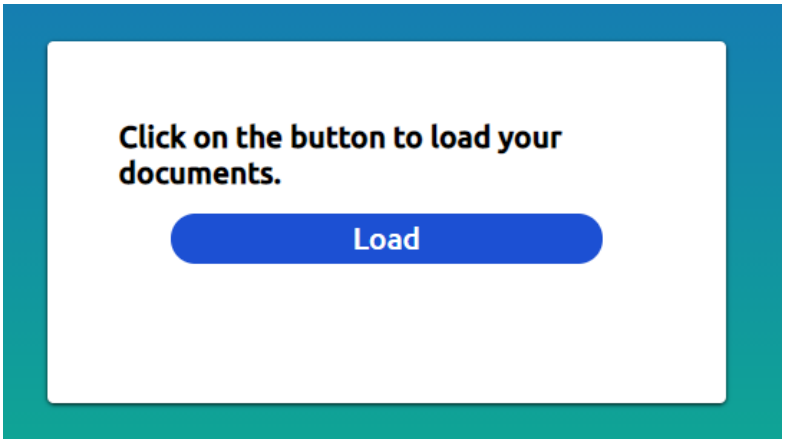


*Figura 40 - Explorar documentos: Carga en la página.*

Cuando el usuario hace clic en el botón, se cargan al instante los documentos que tiene. Estos aparecen como tarjetas con el nombre del documento en grande, el autor, la fecha y el enlace a una puerta de entrada de IPFS para ver el documento. Solo se muestran 3 de estas tarjetas para no ocupar más espacio en la página por lo que se han implementado dos botones para cambiar entre las páginas de tarjetas (figura 41).

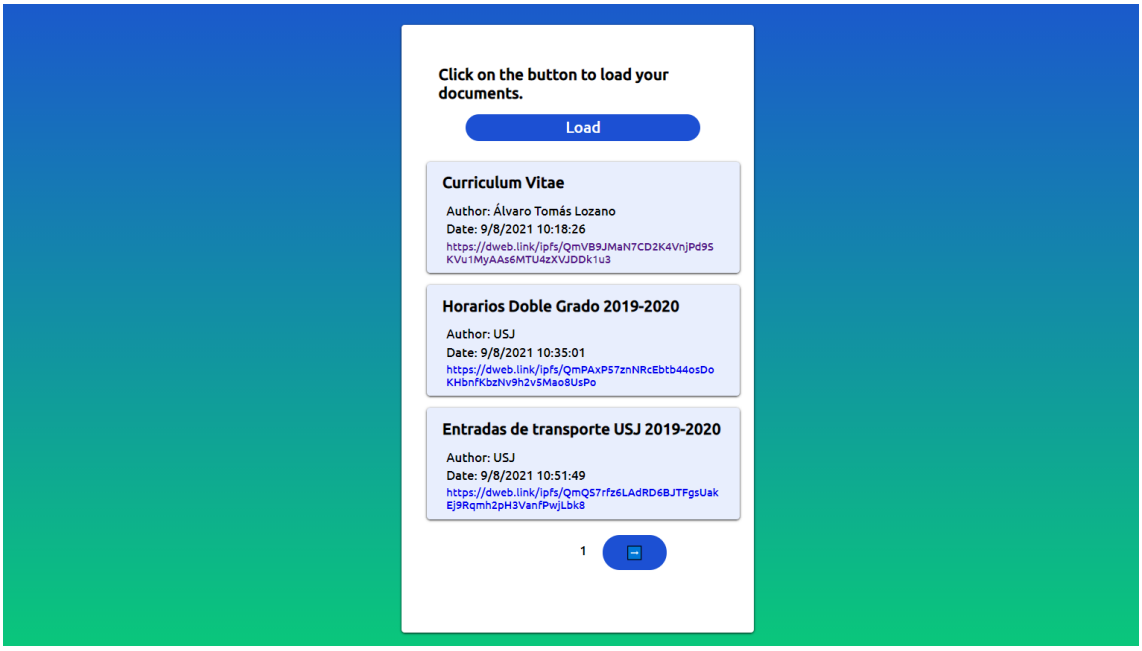


*Figura 41 - Explorar documentos: Tarjetas de los documentos.*

# 6. Estudio económico

En este apartado se hablará del coste que tendría realizar el proyecto en un caso real, con los costes de trabajo en una empresa, personal necesario, costes de los equipos utilizados, software, mantenimiento del servidor web, etc.

Primero, los costes de recursos humanos ya que quizá sea la parte más importante en cuanto al estudio económico. Contando las horas trabajadas de la tabla en la figura 5, encontramos esta información:

- 186 horas trabajadas en el desarrollo de la aplicación y documentación sobre las tecnologías.
- 126 horas trabajadas en escribir y corregir la memoria del proyecto.

Ya que el proyecto ha sido realizado junto a la empresa Inycom, sacando datos del sueldo de un programador junior en la página web de Glassdoor [13] se puede obtener una estimación de los gastos que se necesitarían para pagar el proyecto en base a las horas trabajadas. Para tomar las horas trabajadas se ha tomado 1800 horas de referencia según una aproximación de las horas anuales de trabajo. Se excluyen las horas invertidas en escribir la memoria y las reuniones. Solo se computan las horas trabajadas en el estudio y desarrollo de la aplicación.

Para el sueldo anual, se aplica el 30% de la Seguridad Social. El sueldo base de los programadores junior según Glassdoor es de 19.217 euros y aplicando el incremento de la SS, a la empresa le costaría 24.982 euros. Por lo tanto, en base a la paga por hora que recibe el programador de 10.67 euros, la paga total que recibiría el programador es de 2048,28 euros, mientras que la empresa tendría que pagar 2662,76 euros.

| PUESTO DE TRABAJO | SUELDO ANUAL | COSTE POR HORA | COSTE DEL PROYECTO | COSTE A LA EMPRESA |
|---|---|---|---|---|
| **Program. Junior** | 19.217 € | 10,67 € | 2048,28 € | **2.662,76 €** |

*Tabla 1 – Coste total del proyecto con un Programador Junior.*

En caso de que el proyecto fuese dividido entre el programador que ha desarrollado el apartado de la aplicación de blockchain y luego un *Web Developer* que hiciese una página web más profesional, los costes variarían. También, en Glasdoor [14], aparece el sueldo anual de un programador web y este es de 19.375 euros. Las horas distribuidas serían tal que: el programador junior habría hecho 150 horas y el programador web habría dedicado 36 horas.

| PUESTO DE TRABAJO | SUELDO ANUAL | COSTE POR HORA | COSTE DEL PROYECTO | COSTE A LA EMPRESA |
|---|---|---|---|---|
| **Program. Junior** | 19.217 € | 10,67 € | 1601,42 € | **2.081,84 €** |
| **Web Developer** | 19.375 € | 10,76 € | 387,36€ | **503,56 €** |

*Tabla 2 – Coste total del proyecto con un Web Developer y un programador junior.*

Respecto al coste del hardware empleado, se ha necesitado un portátil para trabajar en el proyecto junto a los periféricos correspondientes. Debido a que se han realizado videollamadas para las reuniones he decidido incluir también los gastos de unos cascos y una cámara web. Estos costes están reflejados en la tabla 3.

| HARDWARE | INVERSIÓN |
|---|---|
| Ordenador Portatil Asus ROG Strix G15 | 1199,99 € |
| Raton Logitech G203 | 27,99 € |
| Webcam Microsoft LifeCam HD 3000 | 24,99 € |
| Cascos Logitech G432 | 39,98 € |
| **COSTE TOTAL** | **1292,95 €** |

*Tabla 3 – Coste del hardware personal para el proyecto.*

Por supuesto, ya que la aplicación web necesita ser alojada para su funcionamiento en un escenario real se tiene que contar con un servidor que pueda tenerla funcional todos los días y además habría que adquirir un dominio personalizado para la aplicación. En la página web de Azure [15] se puede comprobar cuánto costaría alojar una aplicación web en sus servidores. Muestra un desglose de las capacidades y los precios que ofrece en base a donde se elija la ubicación del servidor. La opción más adecuada para los requisitos del proyecto sería el plan

Estándar ya que ofrece un plan enfocado a una aplicación en producción. Para la adquisición del dominio se elegiría Hostalia [16] ya que ofrece una variedad de precios y ajustes para el tipo de dominio que se quiera. Se podría elegir el dominio .com debido a su flexibilidad de cara al cliente y el coste económico que supone. En la tabla 4 hay un desglose de los costes de mantenimiento del servicio anuales y la adquisición del dominio, lo que en total costaría a la empresa 702,28 euros el primer año y 710,78 euros los años consecutivos por el mantenimiento del dominio.

| **SERVICIO** | **INVERSIÓN** |
|---|---|
| Alojamiento en Azure (Estandar) | 58,48€/m – 701,79€/a |
| Compra del dominio | 0,49€/1a – 8,99€/a |
| **COSTE TOTAL ANUAL** | **702,28€ (1er año) - 710,78€/a** |

*Tabla 4 – Coste anual del mantenimiento de la web en funcionamiento.*

Anteriormente se comentaron los gastos que podrían suponer el realizar un contrato inteligente con estado o sin estado y es importante realizar una comparativa de cuánto puede llegar a ahorrar el cliente gracias a la decisión que se realizó. Suponiendo que el coste de 1ETH actualmente es de 2924,48 euros, en la tabla 5 se puede comprobar que realizar el lanzamiento en la red del contrato en su versión original supone un coste de 1,34 euros, mientras que la versión sin estado supone solo 0,41 euros. Este coste está basado en la red de pruebas de Rinkeby por lo que los costes varían extremadamente en la red principal. Lanzar el contrato en su versión *stateful* cuesta aproximadamente 195 dolares, precio que depende del trabajo de los mineros en la red, y su versión *stateless* reduciría su precio hasta los 78,36 dolares. Como se ha dicho, este precio varía con el tiempo en base al precio de la criptomoneda de Ethereum y la disponibilidad de la red. Las transacciones dentro de la red de Rinkeby llegan sin embargo a precios minúsculos, siendo que una transacción del contrato con estado cuesta 0,46 euros y la del contrato sin estado tan apenas 0,08 euros. Este precio será mayor en la red principal, pero en base las pruebas en la red de Rinkeby, la diferencia de precio escala hasta un 575% de diferencia, lo cual indica que la posibilidad de tener un contrato *stateless* reduce enormemente los costes para el usuario final.

| | **Contrato *Stateful*** | **Contrato *Stateless*** |
|---|---|---|
| Lanzamiento del contrato | 0.000461ETH | 0.00014ETH |
| Lanzamiento en Mainnet | 0.056749ETH | 0.022822ETH |
| Transacciones | 0.000158ETH | 0.000028ETH |
| **COSTE LANZAMIENTO** | **1,34€ (R) – 165,5€ (M)** | **0,41€ (R) – 66,37€ (M)** |
| **COSTE TRANSACCIONES** | **0,46€** | **0,08€** |

*Tabla 5 – Coste de un contrato stateful respecto a un contrato stateless.*

Para finalizar, uno de los objetivos del proyecto era analizar los modelos de negocio posibles con la aplicación. El principal modelo de negocio sería cobrar una tasa extra a los usuarios que realizan las transacciones en la red. Ya que la empresa es la que pone parte de los costes para mantener el servidor y el desarrollo de la web, se deberían tener unos costes fijos para los usuarios a parte del coste de la transacción y así lograr un beneficio.

Poniendo de ejemplo los costes que han supuesto el desarrollo actual de la aplicación, nos encontramos con que el desarrollo del primer año ha costado 4657,99€ a la empresa, sumando los costes de desarrollo contando con un programador junior, el coste del hardware y los servicios necesarios para mantener el servidor. Luego, la página costaría 710,78€ al año en cuanto a servicios se refiere, así que mínimo para cubrir los costes de la página del primer año y amortizarla se debe hacer una estimación de cuantos usuarios podría tener la aplicación.

Ya que su utilidad reside en tener una notaría digital, la aplicación le puede ser útil a notarios que requieran de subir sus documentos a un sistema inmutable o administrativos que necesiten sus servicios. Por lo tanto, en el hipotético caso de que anualmente se realizasen 1000 subidas de documentos, el coste de la tasa por transacción para cubrir el gasto inicial debería ser de 4,66 euros. Ya que es un coste elevado para las tasas que se van a realizar, se podría poner un coste de 1,50 euros haciéndolo así más asequible para los usuarios y que acabaría atrayendo a más clientes. Cada año se generarían 1500€ en tasas, lo cual supondría que quedarían por cubrir 3157,99€ y los años posteriores se deducirían del precio 789,22€. Con esta tasa, se tardarían aproximadamente 5 años en cubrir los costes del desarrollo de la aplicación y su mantenimiento.

Otra idea debido a como está estructurada la aplicación es la posibilidad de realizar un modelo de negocio que permita personalizarla a gusto del cliente. Si contamos las horas que ha llevado el trabajo y el coste de personal que ha sido necesitado, se pueden realizar estimaciones de cuánto costaría realizar otro trabajo similar para un cliente u otra empresa que lo necesitase.

Ya que esto daría cabida a un proyecto más profesional, en la tabla 6 se ha plasmado como se considera oportuno que se deberían desglosar los precios de la creación de una aplicación similar, teniendo en cuenta si lo hiciese solo un programador junior o trabajase en conjunto con un *Web Developer*.

Los costes de venta de la aplicación en su estado básico y con modificaciones ligeras del diseño deberían ser similares a los del desarrollo del proyecto en su estado actual. Luego se ofrecería la posibilidad de añadir características extras a coste de 45 euros por hora invertida en la característica. En caso de trabajar con un *Web Developer*, él se encargaría del diseño y se le podría añadir un coste extra por un diseño personalizado. El tener un diseño completamente cambiado de la aplicación web debería ser exclusivo de un desarrollador web por lo que solo se ha incluido en el apartado donde el trabajo sería conjunto. A su vez, al cliente se le cobrarían 50 euros anuales por realizar algún mantenimiento auxiliar de la página en caso de haber algún problema. Los precios finales son personalizables incluyendo el precio base y el plan de mantenimiento.

| **** | **Prog. Junior** | **Prog. Junior + Web Dev.** |
|---|---|---|
| **Precio base de la App** | 2000€ | 2500€ |
| **Plan de Mantenimiento** | 50€/anual | 50€/anual |
| **Características Extra** | 45€/h | 45€/h. + 25€/diseño |
| **Nuevo Diseño** | - | 500€ |
| | | |
| **PRECIO TOTAL BASE** | 2050€+ | 2550€+ |

*Tabla 6 – Coste de personalización y venta de la aplicación web para un cliente externo.*

# 7. Resultados

En esta sección se hablará sobre los objetivos que se han cumplido, la composición final y las características que se han logrado implementar en el proyecto. Los resultados producidos por la aplicación, pruebas y errores detectados durante el desarrollo han sido explicados fundamentalmente en el apartado de desarrollo.

Respecto a los objetivos que tenía el proyecto, estos se dividían en:

- Analizar las diferentes tecnologías Blockchain existentes y su integración con sistemas actuales de identidad digital.
- Definir y desarrollar la arquitectura necesaria para el uso del sistema.
- Diseñar y desarrollar los APIs e interfaces básicos para el manejo e integración del sistema.
- Analizar las posibles oportunidades y modelos de negocio derivados.

Sobre el primer objetivo, no ha habido apenas desviaciones respecto al plan original. Se han estudiado diversas tecnologías de blockchain como Bitcoin para poder entender el funcionamiento de una red de blockchain en primera instancia, aunque la mayor parte del esfuerzo se ha centrado en estudiar la tecnología de Ethereum debido a las grandes ventajas que suponen respecto al resto de tecnologías. La implementación de esta se ha estudiado desde aplicaciones web que simulan juegos utilizando la tecnología de blockchain, hasta subastas donde gracias a la aplicación de contratos inteligentes se permite demostrar la propiedad de las transacciones por venta de objetos. Los conocimientos adquiridos sobre el funcionamiento de la red han sido plasmados en los apartados de introducción y estado del arte de la memoria y en puntos repartidos por el resto de los apartados, por lo cual el objetivo ha sido cumplido.

El segundo objetivo fue cumplido conforme fueron avanzando las reuniones debido a que los conocimientos sobre las herramientas que se podían utilizar para desarrollar una aplicación de blockchain eran escasos. Se brindó ayuda y guía por parte de los tutores para decidir que herramientas utilizar, cómo estructurar el proyecto y su desarrollo, y una vez alcanzado un estado inicial, se logró continuar y tomar las decisiones de diseño necesarias.

Respecto al tercer objetivo, ha sido completado en su totalidad ya que ha supuesto la finalización de una aplicación web que permita a los usuarios subir y ver sus documentos en la red de blockchain, y a su vez, implementar un usuario administrador que funcione como notario para poder ver los documentos que le conciernen.

El cuarto objetivo ha sido comentado en el apartado del estudio económico y mientras que solamente ha sido explicado ahí, realmente las decisiones de diseño han sido influenciadas por un análisis previo a cómo se debía hacer la aplicación para que tuviese salida comercial a futuro. En la primera reunión se habló de cómo el proyecto debía estar enfocado a ser un producto mínimo viable y más adelante, que este sirviese como una solución personalizable para los clientes que quisiesen adquirirlo.

Este proyecto está compuesto por un cliente *dummy* que ha servido como un puente hacia la parte final del proyecto, y por una aplicación web que dispone de todas las funcionalidades necesarias para ser utilizada como un producto comercial. Ambas partes han servido como herramienta de aprendizaje ya que el desarrollo del primer cliente permitió el entendimiento de la tecnología de blockchain y como interactuar con ella, y el segundo ha ayudado a profundizar mejor con el desarrollo web y las aplicaciones que lo rodean.

Finalmente, las características implementadas han sido las necesarias para servir como una aplicación para una notaría digital, en este caso, una web completa y funcional que permita al usuario usar una extensión como puerta de enlace entre la aplicación y la red de blockchain, y que disponga de funcionalidades como subir documentos encriptados a la red, poder obtenerlos de vuelta con una interfaz sencilla y generar a su placer nuevas carteras digitales.

# 8. Conclusiones

Este proyecto fue escogido por curiosidad personal sobre los sistemas de blockchain y las criptomonedas.

El proyecto definitivamente ha alcanzado un estado que puede demostrar las capacidades que tiene la red de blockchain, más como una herramienta que como una moneda. Esta aplicación puede servir como puente hacia el desarrollo de otros tipos de aplicaciones basadas en las mismas tecnologías y lograr obtener un desarrollo personal adecuado. En su estado actual puede ser utilizado por clientes y podría ser lanzada próximamente a un estado de producción, aunque debido a la falta de tiempo, el proyecto requiere de mejoras y trabajo futuro para alcanzar un estado definitivo. Esto es debido a que los conocimientos previos para trabajar en un proyecto de este tipo eran escasos y han tenido que ser adquiridos durante el transcurso del desarrollo.

El tiempo invertido ha sido adecuado, aunque un poco extendido debido al desconocimiento de los campos sobre los que se ha trabajado en este proyecto. Posiblemente con los conocimientos adquiridos se podría realizar un proyecto similar en menos tiempo ya que gran parte de ese tiempo ha sido dedicado a la documentación y formación sobre cómo crear una aplicación basada en Ethereum y como desarrollar una página web desde cero.

Como trabajo a futuro se ha considerado una lista de características que podría tener la aplicación:

- Un buscador de documentos que permita tener una barra de búsqueda y encuentre el documento del usuario por su nombre.
- Diseño de las páginas mejorado.
- Perfil propio cargado desde Metamask donde el usuario pueda interactuar con sus documentos y subirlos a la red, todo desde la misma página.
- Mejora del sistema de notaría para permitir que los notarios tengan funciones más avanzadas.
- Uso de servicios externos para almacenamiento de claves privadas en caso de que los usuarios requieran recuperar su clave.
- Almacenamiento de la clave para encriptar en un servidor seguro.

# 9. Bibliografía

[1] DESCONOCIDO. "LA BLOCKCHAIN. ¿Qué es exactamente la cadena de bloques (Blockchain) y cómo funciona? ¿Qué usos se le pueden dar?", Miethereum, 2017. https://www.miethereum.com/blockchain/

[2] DESCONOCIDO. "Minería Bitcoin ¿Cómo se crea un bloque?", Bit2me Academy. https://academy.bit2me.com/mineria-bitcoin-como-se-crea-un-bloque/

[3] DESCONOCIDO. "¿Qué es Prueba de participación / Proof of Stake (PoS)?", Bit2me Academy. https://academy.bit2me.com/que-es-proof-of-stake-pos/

[4] FRANKENFIELD, JAKE. "Proof of Stake (PoS)", Investopedia, 2021. https://www.investopedia.com/terms/p/proof-stake-pos.asp

[5] AUSTIN, DEREK. "What Are Smart Contracts?", Medium, 2020. https://medium.com/swlh/what-are-smart-contracts-6c13f6c725d7

[6] CONCISE SOFTWARE. "Smart contracts — what are they and how to create them?", Medium, 2019. https://medium.com/@concisesoftware/smart-contracts-what-are-they-and-how-to-create-them-6ddc0df1f66b

[7] WACKEROW. "Gas y tarifas", Ethereum.org, 2021. https://ethereum.org/es/developers/docs/gas/

[8] HOLLANDER, LUIT. "Understanding event logs on the Ethereum blockchain", Medium, 2020. https://medium.com/mycrypto/understanding-event-logs-on-the-ethereum-blockchain-f4ae7ba50378

[9] RODRIGUEZ, ADRIAN. "¿Qué es una Faucet?", Medium, 2020. https://adr-rod87.medium.com/qu%C3%A9-es-una-faucet-7dd256b79bc4

[10] Rinkeby Authenticated Faucet. https://faucet.rinkeby.io/

[11] IPFS Gateway Checker. https://ipfs.github.io/public-gateway-checker/

[12] HARPER, COLIN. "BitcoinPaperWallet 'Back Door' Responsible for Millions in Missing Funds, Research Suggests", Coindesk, 2021. https://www.coindesk.com/tech/2021/02/24/bitcoinpaperwallet-back-door-responsible-for-millions-in-missing-funds-research-suggests/

[13] Glassdoor, "Sueldos para Programador Junior en Inycom". https://www.glassdoor.es/Sueldo/Inycom-Programador-Junior-Sueldos-E2303549_D_KO7,25.htm

[14] Glassdoor, "Sueldos para Web Developer en Inycom" https://www.glassdoor.es/Sueldo/Inycom-Web-Developer-Sueldos-E2303549_D_KO7,20.htm

[15] Azure, "Precios de App Service" https://azure.microsoft.com/es-es/pricing/details/app-service/linux/

[16] Hostalia, "Dominios" https://www.hostalia.com/dominios/

# Anexo I – Propuesta de proyecto

**Nombre alumno**: Álvaro Tomás Lozano

**Titulación**: Graduado en Ingeniería Informática

**Curso académico**: 2020-2021

## 1. TÍTULO DEL PROYECTO

Notario Digital en Blockchain

## 2. DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

El proyecto consiste en al análisis y desarrollo de un sistema de notaría digital basado en Blockchain en el aspecto del aseguramiento e integridad del contenido y firma de documentos notariales por parte de ciudadanos y empresas.

## 3. OBJETIVOS DEL PROYECTO

Los objetivos del proyecto son:

- Analizar las diferentes tecnologías Blockchain existentes y su integración con sistemas actuales de identidad digital.
- Definir y desarrollar la arquitectura necesaria para el uso del sistema.
- Diseñar y desarrollar los APIs e interfaces básicos para el manejo e integración del sistema.
- Analizar las posibles oportunidades y modelos de negocio derivados.

## 4. METODOLOGÍA

La metodología se establecerá en las primeras fases del proyecto.

## 5. PLANIFICACIÓN DE TAREAS

Las tareas quedan predefinidas de manera global en los objetivos. Serán fijadas de forma concreta durante el desarrollo del proyecto.

## 6. OBSERVACIONES ADICIONALES

# Anexo II – Reuniones

| **REUNIÓN: 1** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 09/11/2020 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 10:00 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Hablar de las bases del proyecto.** | |
| 2 | **Explicaciones sobre la notaría digital y el blockchain.** | 001 |
| 3 | **Establecer la metodología a seguir para el proyecto.** | |
| 4 | **Hablar sobre el estado final del proyecto como Producto Mínimo Viable y la salida comercial.** | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Explicar qué es el blockchain, el Proof of Work, el Proof of Stake, y cómo funciona una cartera digital. | Siguiente Reunión | Álvaro Tomás |
| 002 | | | |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 2** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 23/11/2020 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 9:30 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Corregir las explicaciones sobre lo estudiado para la reunión.** | |
| 2 | **Hablar sobre el funcionamiento de los nodos en blockchain.** | |
| 3 | **Tratar el uso de un sistema de red local de Ethereum con Ganache** | 001 |
| 4 | **Hablar sobre cómo funciona un Smart Contract y como lanzar uno.** | 002 |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Instalar y realizar pruebas en la red de Ganache. | Siguiente Reunión | Álvaro Tomás |
| 002 | Crear un Smart Contract sencillo y lanzarlo en Ganache. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 3** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 09/12/2020 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 9:47 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar los avances realizados en la red local de Ethereum.** | |
| 2 | **Mostrar el uso de una cartera digital que interactúa con la red.** | |
| 3 | **Enseñar el Smart Contract y su funcionamiento.** | |
| 4 | **Hablar sobre el funcionamiento de Remix para lanzar Smart Contracts.** | |
| 5 | **Hablar sobre IPFS, Solidity y su funcionamiento.** | 001 |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Documentarse sobre IPFS y crear un Smart Contract más avanzado. | Siguiente Reunión | Álvaro Tomás |
| 002 | | | |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 4** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 22/12/2020 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 9:55 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar los avances sobre IPFS y el Smart Contract.** | |
| 2 | **Proponer el cambio de la red local a una red pública de Ethereum.** | 001 |
| 3 | **Explicar el funcionamiento del faucet para asignar Ethers.** | 002 |
| 4 | **Recomendar la página de Infura para obtener un nodo.** | 003 |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Conectar el Smart Contract con la red de Rinkeby y lanzarlo. | Siguiente Reunión | Álvaro Tomás |
| 002 | Aprender a asignar fondos a las carteras con faucet. | Siguiente Reunión | Álvaro Tomás |
| 003 | Registrar una cuenta en Infura y crear un proyecto para obtener un nodo. | Siguiente Reunión | Álvaro Tomás |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 5** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 04/05/2021 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 10:00 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar las metas para la anterior reunión.** | |
| 2 | **Comentar los problemas que hubo para ello y analizar la forma de solucionarlo.** | 001 |
| 3 | **Enseñar el prototipo de cliente base a falta de ser funcional.** | |
| 4 | **Repasar las funciones que debe tener el cliente.** | 002 |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Repasar como se conecta el contrato a Rinkeby y conseguir implementarlo en el cliente. | Siguiente Reunión | Álvaro Tomás |
| 002 | Conseguir la conexión con el nodo de Infura y probar a mandar documentos. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 6** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 01/06/2021 | | |
| **Hora comienzo:** | 9:00 | **Hora finalización:** | 9:30 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar el progreso de la conexión con la red de Rinkeby y el Smart Contract.** | |
| 2 | **Enseñar como se consigue un hash de IPFS en el cliente base.** | |
| 3 | **Proponer como cambiar las llamadas desde el cliente con el Smart Contract.** | 001 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Conseguir realizar llamadas al Smart Contract desde el cliente base y que haga una transacción en la red de pruebas. | Siguiente Reunión | Álvaro Tomás |
| 002 | | | |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 7** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 08/06/2021 | | |
| **Hora comienzo:** | 8:30 | **Hora finalización:** | 9:04 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar las llamadas a la red de pruebas.** | |
| 2 | **Sugerir cambios para que el cliente pueda conectarse a la aplicación desde Metamask.** | 001 |
| 3 | **Analizar los cambios de las llamadas respecto a las funciones disponibles de Web3JS.** | 002 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Hacer que el cliente pueda enlazar la cartera con la aplicación sin necesidad de su clave privada. | Siguiente Reunión | Álvaro Tomás |
| 002 | Probar a utilizar el método send() del contrato para crear la transacción. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 8** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 15/06/2021 | | |
| **Hora comienzo:** | 8:30 | **Hora finalización:** | 9:10 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revisar los avances con la aplicación.** | |
| 2 | **Sugerir la implementación de una función que calcule el gas y el cifrado del documento.** | 001 |
| 3 | **Explicar como funcionan los generadores de carteras en internet.** | 002 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Obtener el gas necesario para lo que se va a subir y cifrar el hash de IPFS junto a los metadatos. | Siguiente Reunión | Álvaro Tomás |
| 002 | Crear un generador de carteras sencillo. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 9** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 02/07/2021 | | |
| **Hora comienzo:** | 8:00 | **Hora finalización:** | 8:40 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revision del estado actual del Smart Contract.** | |
| 2 | **Revision de la función para calcular el gas en la aplicación.** | 001 |
| 3 | **Proponer un contrato stateless para reducir los costes.** | 002 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Mejorar la función de calculo de gas ya que no funciona correctamente. | Siguiente Reunión | Álvaro Tomás |
| 002 | Investigar sobre los contratos stateless y probar a implementar uno. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 10** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 08/07/2021 | | |
| **Hora comienzo:** | 12:00 | **Hora finalización:** | 12:40 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Revision del estado del generador de carteras.** | |
| 2 | **Valoración del nuevo contrato de forma stateless.** | |
| 3 | **Explicar como debe funcionar la búsqueda de los documentos.** | 001 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Crear un apartado que permita obtener los documentos subidos en la aplicación. | Siguiente Reunión | Álvaro Tomás |
| 002 | | | |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 11** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 15/07/2021 | | |
| **Hora comienzo:** | 10:30 | **Hora finalización:** | 11:25 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Proponer el comienzo del desarrollo de la web final.** | |
| 2 | **Poner de ejemplo webs con un diseño estilizado y simple.** | |
| 3 | **Explicar las características finales que debe tener la web.** | 001 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Comenzar con el desarrollo de la web final. | Siguiente Reunión | Álvaro Tomás |
| 002 | | | |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 12** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 29/07/2021 | | |
| **Hora comienzo:** | 10:30 | **Hora finalización:** | 10:55 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Mostrar el diseño actual de la web y el progreso.** | |
| 2 | **Proponer como deben ser las características en la web visualmente.** | 001 |
| 3 | **Dar consejos sobre los apartados de estado del arte y estudio económico de la memoria.** | 002 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Finalizar la web con lo implementado en el cliente dummy. | Siguiente Reunión | Álvaro Tomás |
| 002 | Hacer la memoria e incluir las actas. | Siguiente Reunión | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |

| **REUNIÓN: 13** |
|---|

| | | | |
|---|---|---|---|
| **Fecha:** | 08/09/2021 | | |
| **Hora comienzo:** | 10:30 | **Hora finalización:** | 11:20 |
| **Lugar:** | Remoto desde casa | | |
| **Elabora acta:** | Álvaro Tomás | | |
| **Convocados:** | Álvaro Tomás, Rodrigo Casamayor, Miguel Ángel Barea | | |

**Orden del día / Acta**

| No. | Asunto | Acuerdo |
|---|---|---|
| 1 | **Mostrar la pagina web final y todas sus características.** | |
| 2 | **Enseñar los apartados de la memoria y lo que queda por corregir.** | 001 |
| 3 | **Explicar como se han abordado ciertos temas en base a lo comentado en las reuniones.** | |
| 4 | **Dar el visto bueno al proyecto y comentar sugerencias para la presentación.** | 002 |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |

*Resumen de acuerdos*

| Número | Acuerdo | Plazo | Responsable |
|---|---|---|---|
| 001 | Terminar de corregir la memoria y entregarla | 09/09/2021 | Álvaro Tomás |
| 002 | Preparar la presentación. | 20/09/2021 | Álvaro Tomás |
| 003 | | | |
| 004 | | | |
| 005 | | | |
| 006 | | | |