Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en tecnologías de la información

ADAPTACIÓN E IMPLANTACIÓN DE UNA ARQUITECTURA BLOCKCHAIN PARA DAR SOPORTE AL TRABAJO DE EQUIPOS EN UNA ORGANIZACIÓN

Alumno: Saúl Llamas García

Tutores: Joaquín Nicolás Adiego Rodriguez
Natalia Martín Cruz

A mis padres que me han dado todo.

Agradecimientos

A mi padre, por haberme todos los recursos que me han permitido llegar hasta aquí.

A mi madre, por ser una luchadora que ha cuidado toda la vida de mí.

A mis hermanos que han estado ahí siempre que los necesito.

A mis amigos que siempre han estado ahí en los buenos y en los malos momentos.

A la gran cantidad de compañeros con los que he tenido honor de compartir clase, y que han logrado que este camino haya sido mucho más fácil de lo que parecía.

Resumen

El proyecto expuesto en este documento tiene como objetivo desarrollar una aplicación que permita dar soporte a la planificación, gestión y validación de entregas y tareas de los grupos de trabajo que forman una organización, haciendo uso de las tecnologías emergentes de la *blockchain* con el objetivo de mantener un registro inmutable de todas estas actividades.

Este proyecto ha sido desarrollado sobre una red Ethereum, haciendo uso de la implementación Geth (Go Ethereum) para el despliegue de esta. Así como se ha utilizado también el sistema IPFS (Sistema de archivos interplanetario) para el almacenamiento de archivos.

Para el desarrollo de contratos inteligentes se ha utilizado el lenguaje Solidity.
En cuanto al desarrollo del *front-end* ha sido realizado en JavaScript, haciendo uso de la librería React, así como se han utilizado las tecnologías HTML5 y CSS. Para la comunicación con la red Ethereum se ha utilizado la librería Web3JS.

Abstract

The exposed project presented in this document aims to develop an application which allows to support the management and validation of deliveries and tasks of the working groups which form an organization. This project makes use of emerging blockchain technologies, adapting the traditional architecture of an application to these new technologies

This project has been developed on an Ethereum network, making use of the Geth implementation (Go Ethereum) for its deployment as well as IPFS (Interplanetary File System) system for file storage.

The Solidity language has been used for the development of smart contracts. Regarding the front-end development, it has been performed in JavaScript, using the React library, as well as HTML5 and CSS technologies. For communication with the Ethereum network, the Web3JS library has been used.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Cuando escuchamos "blockchain"lo primero que se nos pasa por la cabeza son las criptomonédas, pero hay mucho más detrás de esta tecnología.(Explicada en detalle en el capítulo 2).En los últimos tiempos se está utilizando esta tecnología para tareas tales como procesos de licitación de contratos [68], alternativas de DNS [47], o aplicaciones descentralizadas (DApps) [8].

En esta nueva rama del desarrollo de aplicaciones distribuidas (DApps) se engloba este proyecto que toma la idea base de los algoritmos que utilizan las tecnologías de la *blockhain*, con el objetivo de realizar una aplicación que gestione conjunto de tareas que se producen en el flujo del trabajo en equipo una organización, manteniendo así un registro inmutable de los datos asociados a las entregas de esas tareas.

A groso modo, estos algoritmos (Explicados en detalle en la sección 2.1.2) de los que se toma la idea de proyecto, permiten que todos los miembros de la red sincronicen un conjunto de información que todos tienen en común, que es inalterable. Y que cada inserción que se produzca en este conjunto de información, se realice con la seguridad de que cumple con las condiciones anteriormente expuestas, y que por tanto pueda ser sincronizada de forma segura por el resto.

De esta manera el conjunto de personas que forman un grupo dentro de un proyecto de una organización , que tienen unas tareas asignadas, podrán realizar una entrega asociada a esa tarea. Este proceso de entrega consta de un proceso de consenso del contenido de la entrega, donde todos los miembros dan su 'sí' explicito a que el contenido de esta es el correcto y final, y que por tanto esta entrega sea almacenada de manera inalterable en un registro histórico e inmutable, es decir, en la *blockchain*.

1.1. Motivación

La idea de este proyecto nace de los algoritmos de la *blockchain* (Explicados en detalle en la sección 2.1.2), que logran que todos sus miembros mantengan un conjunto de datos inalterables de forma común, sin necesidad de que una entidad centralizada intervenga en la validación de la introducción de nuevos datos.
De esta forma, estos mecanismos logran alcanzar un consenso entre los miembros de la red, sobre la sobre la integridad e inmutabilidad de los datos que residen sin necesidad de recurrir a un tercero.

Se toma por tanto esta idea a para lograr que un grupo de trabajo formado por diversos miembros, realice una validación en conjunto de la entrega, y que cuando esta tenga el consenso de todos los miembros, sea almacenada en la *blockchain*.

De esta manera, se producirá un proceso de consenso donde mediante la aplicación cada miembro del grupo deba dar un 'sí' explícito, respecto a que este es el resultado final de una entrega asociada a una tarea. De esta manera, el grupo logra llega a un consenso de que este es el resultado final. Por tanto una vez finalizado el consenso esto quedará plasmado dentro de la *blockchain* (Explicado en el capítulo 2), pues como cada miembro será participe de la red, cuando se produzca el registro en la *blockchain* de esta entrega por parte de uno de los miembros, al ser todos participes de una red de consenso, esta información quedará almacenada de forma inmutable e accesible para todos estos miembros, siendo implícitamente aceptada por todos los participantes de la red.

1.2. Objetivos

El objetivo general de este proyecto es desarrollar una aplicación de trabajo en equipo basada en una arquitectura que tome por tecnología de almacenamiento principal la cadena de bloques, y que permita mantener en esta un historial inmutable de las tareas realizadas y de su contenido.

1.2.1. Objetivos técnicos de desarrollo

Los objetivos principales de este proyecto son:

- Análisis de las tecnologías existentes con el objetivo de seleccionar una segura y acorde a la idea del trabajo en equipo.
- Adecuación de la aplicación a las retracciones impuestas por las tecnologías de la *blockchain* que surjan durante el desarrollo del proyecto, con el objetivo de mantener la idea base de la aplicación.
- Proporcionar un mecanismo de validación de tareas en equipo, que permita mantener un registro inmutable de las transacciones (Entregas) de los usuarios

- Proporcionar un acceso a la aplicación por parte del usuario sin requerir de software de terceros.

1.2.2. Objetivos personales

En cuanto a mis objetivos personales basados en la finalización de mis estudios residen:

- Conocer un nuevo *framework* de desarrollo web, mejorando así mis facetas del desarrollo web en el apartado de *front end*
- Conocer nuevas tecnologías futuras alternativas de *backend* como pudiera ser la *blockchain*.
- Complementar la planificación y desarrollo de un proyecto, con los realizados en mis prácticas curriculares con el objetivo de formarme acerca de diversas metodologías o mejorar en ellas.
- Ganar independencia en el trabajo buscando retos en tecnologías emergentes, donde encontraré nuevos retos y problemas a solventar.
- Aprender nuevos lenguajes de programación como Solidity.
- Realizar un despliegue real de una aplicación.

1.3. Usuarios objetivos

Principalmente esta aplicación esta destinada a ser utilizada como un sistema de validación dentro del ámbito escolar por parte de mis tutores de TFG. Sería por tanto utilizada para la entrega y validación de tareas dentro de grupos de trabajo en aula.

Adicionalmente,esta aplicación tiene también como mercado secundario a aquellas organizaciones que necesiten, o tengan pensado en un futuro utilizar un sistema de organización y planificación de tareas, que requiera de una de validación segura, previa a la entrega por parte de los grupos de trabajo presentes en la organización. Utilizarían esta aplicación en un entorno de red privado.

1.4. Estructuración de la memoria

El presente documento está estructurado en los siguientes capítulos:

- **Capítulo 2 Estado del arte**, este capítulo recoge la investigación previa de las tecnologías existentes *blockchain*, así como el funcionamiento de esta.

- **Capítulo 3 Planificación y desarrollo del proyecto**, en este capítulo se aborda la planificación inicial realizada en el proyecto, aplicando una metodología ágil.
- **Capítulo 4 Plan de riesgos y costes**, en este capítulo se realiza una estimación del coste del proyecto y de los posibles riesgos a los que se enfrenta este proyecto.
- **Capítulo 5 Análisis previo**, en el marco de la metodología ágil, este capítulo está destinado a el análisis previo que se realiza en el sprint 0 con el objetivo de esclarecer una visión más clara del proyecto.
- **Capítulo 6 Tecnologías utilizadas**, capítulo destinado a la descripción de las tecnologías seleccionadas en este proyecto.
- **Capítulo 7 Diseño**, capítulo destinado a la fase de diseño de la aplicación
- **Capítulo 8 Implementación, despliegue y pruebas**, capítulo destinado a la fase de implementación y despliegue de la aplicación, y las pruebas realizadas sobre este
- **Capítulo 9 Seguimiento del proyecto**, este capítulo está destinado a la serie de sprints que se han realizado a lo largo del proyecto, y los eventos relacionados con estos.
- **Capítulo 10 Conclusiones**, capítulo final para realizar una valoración del proyecto, y el posible futuro de este proyecto.

Finalmente existen una serie de apéndices donde está el manual de usuario

Capítulo 2

La blockchain

2.1. Fundamentos de la blockchain

2.1.1. Introdución a la blockchain

Una cadena de bloques (en inglés *blockchain*), es una estructura de datos cuya información se agrupa en conjuntos (bloques) incorporando metadatos, relativas a otro bloque de la cadena anterior en una línea temporal. Debido a las técnicas criptográficas empleadas, la modificación de un bloque solo puede ser lograda modificando todos los bloques anteriores, que por tanto a medida que crece logra un almacenamiento de información irrefutable e inmutable. Podría verse por tanto como una base de datos no relacional (un *log* o un registro histórico) donde se almacena información relativa a transacciones que han ocurrido en un sistema o entre usuarios.

2.1.2. Principios de la blockchain

La gobernanza es una estructura a través de la cual un participante o usuario de un sistema acepta utilizar este bajo una serie de normas. Para que cualquier sistema de gobernanza funcione correctamente, deben existir tres elementos sin que estos interfieran entre sí: [21]

- Gobernantes
- Reglas
- Participantes

Una de las características clave de *blockchain* es la descentralización, incrementando la complejidad de de la gobernanza sobre *blockchain*. Por tanto aquí es donde entra el concepto

de reglas en forma de algoritmos de consenso que permiten la verificación de transacciones dentro de la red.(Ver sección 2.1.2) Por tanto la implementación de estos algoritmos, recae a cargo de los diferentes desarrolladores que son una de las piezas clave en cuanto a logran la confianza de los participantes, y su impacto en ellos, y en la propia cadena de bloques.

Un participante por tanto es cualquier persona que dispone de un nodo conectado a una red *blockchain* (Tiene una dirección pública en código *hash* asociada a una clave privada). Dentro de estos participantes, nos encontramos con el concepto de minero. Un minero se encarga de validar nuevas transacciones y grabar los datos correspondientes en la *blockchain*. El proceso de minar requiere de un alto costo de computación (Haciendo uso de la GPU y CPU y que tienen detrás un gasto energético), y permite agrupar una serie de transacciones en un bloque. El minero comprueba que estas transacciones no estén en ningún otro bloque a la hora de validad, de ahí su alto coste de cómputo, y en caso de estar ya validada se descarta.

Entenderemos entonces que un bloque es un registro que permite almacenar o mantener un registro de las transacciones que se han producido. Una transacción puede ser el envío de dinero a otra cuenta, o la ejecución de un *smart contract*. (Ver sección 2.1.3)

Una vez que el minero ha formado su propio bloque, debe firmar este. Para lograr esta firma debe resolver un algoritmo matemático de computación que tendrá como resultado un *hash*. De esta manera, no nos encontraremos dos bloques con la misma firma.
Este proceso es bastante tedioso, pues el resto de mineros compiten por encontrar un *hash* válido con el que firmar su bloque, entrando además en juego el término de "dificultad de minado". [1]

La dificultad en la *blockchain* juega un papel muy importante. En primer lugar, regula la fabricación y activación de la criptomoneda a un ritmo constante y predeterminado. En segundo lugar, permite mantener el tiempo de producción de bloques de la criptomoneda de forma uniforme evitando problemas de seguridad y evitando posibles fraudes. [2]

El problema de los mineros no acaba aquí, una vez que han minado dicho bloque, este es transmitido y validado por el resto de nodos. Es decir, todos los comprueban que ese bloque cumple con las condiciones impuestas, y entonces se produce la confirmación del bloque e inserción de bloque. Al insertarse este último bloque, todos aquellos mineros que tomaron al anterior bloque como referencia, generarán un *hash* erróneo, y por tanto acabará siendo descartado, y por tanto generándose un gran derroche de energía eléctrica utilizada en el hardware que permite realizar esto cálculos.

Si finalmente este proceso culmina satisfactoriamente el minero se verá recompensado, en función del tipo de moneda que rija en esa red.La recompensa de bloque es el origen de todas y cada una de las criptomonedas dentro de una *blockchain*. De hecho, es la única forma de generarlas y convertirlas en parte del sistema económico que sostiene una *blockchain*. Su objetivo es "pagar.ªl minero por el trabajo realizado. [7]

Esta recompensa generada es almacenada en la cartera ò monedero. (En inglés *blockchain wallet*). Estas recompensas por tanto como hemos dicho anteriormente, hacer referencia a las criptomonedas, y se pueden realizar como moneda de compra en aquellos sitios que lo regulen.

Algoritmos de consensuación

Un algoritmo de consenso es el mecanismo que permite seleccionar el estado correcto de un registro después de realizar una transacción. El resultado de este algoritmo por tanto indica que es una verdad absoluta que todos los nodos deben seguir, de ahí el concepto de reglas que hablamos anteriormente, y que permiten el funcionamiento correcto de la *blockchain*.

Entre los algoritmos existentes los más utilizados son: [25]

- *Proof of Work (PoW)*: Las propuestas de validación de bloques llegan con el resultado de un cálculo complejo de *hash* que requiere bastante potencia de cálculo. Debido a la complejidad de cálculo, y la potencia requerida para calcular este (Pues la potencia es medida en la cantidad de energía eléctrica que es necesaria para dar soporte al hardware que permite realizar los cálculos), se admite que no se enviará un bloque falso. Además la verificación del cálculo es bastante rápida y sencilla y por tanto es fácil determinar que este era falso, evitando así posibles intentos de falsificación.
  PoW permite asegurar que el conjunto de bloque es inmutable. Entre sus inconvenientes, residen la cantidad de cálculos que son desperdiciados (Electricidad que se requiere para estos cálculos). A medida que la red crece, esta es más resistente a un ataque del 51 %. [9]
- *Proof of Authority(PoA)*: Es el algoritmo elegido por muchas de las redes privadas, porque para poder participar en dicha red es necesario tener que ser aceptado, y por tanto este miembro ya no es anónimo (Pérdida de confidencialidad). Cada participante de la red firma las operaciones con su certificado digital, de manera que el minero solo valida la cabecera y no recibe una recompensa, si no que se muestra como confiable. Por tanto su uso está limitado solo a redes privadas, y básicamente no existe minado con el objetivo de obtener una criptomoneda como recompensa, ni gasto para enviar una transacción.
- *Proof of Stakes*: Es un algoritmo que basa la confianza en un participante en la cantidad de trabajo que ha realizado en la validación de un bloque. La inversión en computación se transforma en fiabilidad y confianza en ese participante. El problema reside en que este algoritmo puede recaer en una centralización en los miembros que validan los bloques.

2.1.3. Diferencias entre smart contract y la firma digital

Primariamente es necesario introducir un nuevo concepto, *"smart contract"*. Un *smart contract*, es un contrato que tiene validez sin depender de una autoridad centralizada, que es público y accesible para cualquier persona, y está alojado en la *blockchain*. Este código visible puede ser ejecutado, y es inmutable y transparente una vez que está almacenado en un bloque. Un *smart contract* tiene asociado una dirección pública. (Address)

La idea de un *smart contract* es la de poder tener contratos ejecutables e inmutables, que serán ejecutados por un programa informático donde utiliza los acuerdos establecidos entre

dos o más partes en dicho contrato, dando por tanto un resultado que cumple una serie de condiciones específicas.

En la sección 2.1.2 vimos que el algoritmo *PoA* utilizaba certificados digitales, y basaba la validación de los bloques en este. Existe una relación entre ambos conceptos, certificados digitales y *smart contracts*. Ambos logran confirmar la identidad de una de las partes o totalmente de ambas. Sin embardo la principal diferencia reside como hemos visto, en que el certificado pertenece a una de las partes. Para lograr la totalidad de confianza de las partes es necesario que todos cuenten con un certificado. Este certificado es obtenido **de una autoridad centralizada**, contraria a la descentralización que busca la *blockchain*.

Sin embargo en el caso de los *smart contracts*, son desplegados en la *blockchain*, y no requiere que una autoridad centralizada valide este, si no que esto es logrado gracias a los algoritmos de consenso. La autenticidad se logra debido a que ambas partes pertenecen a una red de confianza donde mediante los algoritmos han llegado a un estado de confiar el uno en el otro, y que tras un proceso de computación bastante costoso este contrato ha sido desplegado, es público, e inmutable. En el caso del algoritmo PoA que utilizaba los certificados, la confianza se gana minando, ya que no se recibe ninguna recompensa, y ese esfuerzo se transforma en confianza frente al resto de nodos.

Ambos mecanismos se basan en las técnicas criptográficas, y como hemos visto su diferencia reside en la centralidad de la autorización y en la forma de verificar las identidades. De ambas formas podríamos llegar a la confianza del contrato, en el caso de los certificados todos firmarían ese contrato gracias a su certificado digital, y en el caso de los *smart contract*, la confianza reside en la propia red y en el despliegue de ese contrato.

2.2. Análisis de las redes y tecnologías blockchain actuales

Se ha realizado un análisis previo de las actuales redes *blockchain* para el desarrollo de aplicaciones basadas en *blockchain*. Todas comparten en común que son redes de código abierto. [14] [61]

- **Ethereum**. Tiene a su favor con que es la red más conocida, y activa actualmente. Cuenta con una gran variedad de herramientas de desarrollo y funcionalidad reutilizable. Es la red con mayor mantenimiento, dispone más nodos que *BitCoin* actualmente, y una mayor comunidad de usuarios. La cantidad de usuarios repercute en la inmutabilidad de los bloques, y por tanto es la red más confiable actualmente.

  Ethereum por contra tiene una baja velocidad de transacciones, cuenta con un alto coste de despliegue y por transacción.

- **NEAR** Cuenta con una alta escalabilidad de nodos, con bajos costos de transacción y interfaz que permite el desarrollo rápido de DApps.

Sin embargo solo dispone de una red principal, que es bastante joven. Sus desarrolladores cuentan una menor experiencia frente a otras redes.Su mayor problema es la gran falta de manuales y de documentación.

- **EOS** Entre sus ventajas podemos encontrar la alta velocidad de transacción rápida, sin tarifas de transacción. Además permite en desarrollo de algoritmos complejos y la paralelización

  Entre sus principales inconvenientes se encuentra su sistema validación, cuya complejidad es bastante alta, incluso para desarrolladores experimentados.Gran complejidad en la creación de cuentas, y falta de validadores (Mineros) en la red.

- **SOLANA** Destaca su sistema de transacciones, la escalabilidad de la red y que permite la paralelización de transacciones. Permite una gran variedad de lenguajes para el desarrollo de *Smart Contracts*.

  Sin embargo tiene muy poca transparencia en el mecanismo de consenso.Existe una avaricia de solicitud de recursos en los nodos. Además cuenta con una de las más pequeñas comunidades de usuarios.

- **Hyperledger** No tiene red pública, está pensado para el desarrollo de aplicaciones,pero su principal problema reside en la falta de escalabilidad, problemas en el almacenamiento de datos, y requiere una alto mantenimiento de la red. Debido a problemas de seguridad, la mayoría de proyectos han debido abandonar esta tecnología.

- **Hedera Graph** Tiene la capacidad de manejar cientos de miles de transacciones por segundo y autenticar más de un millón de firmas por segundo ofreciendo un alto nivel de seguridad

- **Quorum** Ofrece privacidad a nivel de transacciones, junto con transparencia en toda la red. No es compatible con la política global de intercambio de datos, y su objetivo de usuario está orientado al sector bancario

2.3. Análisis de aplicaciones basadas en Ethereum

2.3.1. Proveedores

Antes de hacer un análisis, cabe decir que para conectar la *blockchain* con la web convencional es necesario de un proveedor. Un proveedor es una herramienta que permite hacer uso de las tecnologías *blockchain* en el navegador, interconectando así el navegador con el nodo Ethereum donde está situada la cuenta. Este nodo en el caso de Ethereum puede administrar varias cuentas.

La mayoría de aplicaciones basadas en Ethereum permiten el uso de Metamask. Metamask es una extensión web de navegador que permite desplegar un nodo ligero (Un nodo ligero

quiere decir, que este no sincroniza la totalidad de los bloques de la *blockchain*, si no una cantidad reducida de estos.) en el navegador, sin la necesidad de tener que desplegar un nodo completo en el equipo y configurarlo. Además gestiona la cartera del nodo (Blockchain Wallet), así como brinda seguridad en las transacciones realizadas por los usuarios.

2.3.2. Aplicaciones existentes basadas en Ethereum

Entre las aplicaciones más famosas nos podemos encontrar: [8]

- **Peepth**: Es la red social referente en el mundo de la *blockchain*, que calca el funcionamiento de Twitter pero de forma descentralizada. Para evitar problemas de privacidad almacena las publicaciones en IPFS (Un sistema de almacenamiento basado en hashes) permitiendo de esta manera la eliminación del contenido de estas, pues cabe recordar que el contenido grabado en los bloques de *blockchains* como Ethereum es inmutable y no puede ser borrado. [51]
- **Gnosis**: Implementa un sistema de votación de predicción. Se puede votar desde el tiempo que hará en un día determinado de la semana, hasta los posibles resultados de unas elecciones. De esta manera se logra una mayor veracidad de los resultados de la votación, pues una vez que este se produce no puede ser modificado.
- **EtherRol**: Juego de apuesta basado en el resultado de un dado.
- **Livepeer**: Es una aplicación de retransmisión en vivo.
- **Brave Frontier Heroes:** Es un juego online vinculado a una cuenta Ethereum, y que permite compras in-game mediante dicha criptomoneda.
- **Catalog**: Es un sistema de retransmisión , intercambio, y descarga de canciones.

2.4. La importancia de la blockchain en este proyecto

Por tanto, la importancia de la *blockchain* frente a otros proyectos que permitan la gestión de tareas o entregas, como por ejemplo la platarforma de Campus Virtual de la Universidad de Valladolid, es que esta permite mantener un registro inmutable de estas entregas.

En una aplicación convencional, no existe la certeza de que estos datos no puedan ser modificados por terceros. En una aplicación basada en *blockchain*, debido a los principios que rigen en esta, tenemos la certeza de que los datos almacenados son inmutables y seguros.

Capítulo 3

Planificación y desarrollo del proyecto

3.1. Enfoque seleccionado

Debido a la incertidumbre provocada por las restricciones de las tecnologías emergentes de la *blockchain*, ya que no conocemos los inconvenientes que nos encontraremos a la hora de adaptar e implementar esta tecnología a una arquitectura de aplicación común, no podemos determinar con facilidad los requisitos y la funcionalidad a la que se podrá dar soporte. Además la variedad de tecnologías nuevas que he de aprender, donde muchas de ellas tienen una curva de aprendizaje alta, reside otra dificultad en este proyecto debido al corto periodo de tiempo en el que está pensado realizarse.

De esta forma, se ha decidido seleccionar una metodología ágil para poder dar respuesta a la incertidumbre presente en los requisitos, que posiblemente variarán a lo largo del proyecto en función del los problemas que vayamos encontrando, incorporando y eliminando nuevos requisitos a lo largo del proyecto, y de esta manera poder realizar a tiempo el proyecto.

Una metodología ágil, es un enfoque para la toma de decisiones en los proyectos de software, referido a los métodos de ingeniería del software basados un desarrollo iterativo y de forma incremental, donde los requisitos y soluciones varían a lo largo del tiempo en función de las necesidades.

Estas metodologías se basan en una serie de iteraciones sobre un ciclo de vida cuyas fases son: la planificación, el análisis de requisitos, diseño, codificación, pruebas y documentación. El objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona"(sin errores). [19]

3.1.1. Scrum

Se ha seleccionado el *framework* Scrum para el desarrollo de este proyecto.

Scrum es un proceso de gestión que reduce la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes. La gerencia y los equipos de Scrum trabajan juntos alrededor de requisitos y tecnologías para entregar productos funcionando de manera incremental usando el empirismo.

Existen tres pilares básicos que soportan toda la implementación del control de procesos empíricos la transparencia, la inspección y la adaptación. [4] [34]

- La transparencia: Debe emplearse una terminología regida por un estándar común, de tal forma que los observadores puedan comprender y entender lo que están viendo, así como todos estos aspectos deben ser visibles para los interesados en el resultado.
- La inspección: Aquellos usuarios involucrados en el marco de Scrum, deben realizar una inspección del progreso realizado con el objetivo de evitar variaciones indeseadas. La frecuencia de estas no debe ser alta, para evitar que interfiera en el trabajo.
- La adaptación: En caso de producirse una variación por encima de los límites marcados como aceptables, debe reajustarse el proceso o material que estaba siendo procesado. Dicho ajuste debe producirse lo más pronto que sea posible con el objetivo de minimizar el crecimiento de esa desviación.

3.1.2. Organización de equipos y roles en Scrum

Los grupos de trabajo de Scrum (*"Scrum Teams"*) están formados generalmente de 3 a 10 miembros, es decir, que el tamaño de estos suele ser bastante pequeño. Dentro de estos grupos de trabajo existen tres roles (Que podemos visualizarlos en la figura 3.1 entablados dentro de la categoría de "roles centrales".La otra categoría sería la de roles no centrales). Estos tres roles son: [34]

- ***Product owner***: El Dueño de Producto es el responsable de maximizar el valor del producto y el trabajo del Equipo de Desarrollo(Este proceso podría variar en cada organización). Además es el responsable de gestionar la lista de producto. (Ver *Product Backlog* en la sección 3.1.4)
- ***Development Team***: Formado por el grupo de personas que realizan la creación de incremento (Ver incremento en la sección 3.1.4), así como realizan la labor de entregar un incremento de producto "terminado", que pueda ser potencialmente desplegado en producción al final de cada "sprint"(Ver Sprint en la sección.3.1.3). Cabe decir que estos grupos son autoorganizados, pues no existe una orden de un miembro superior de como transformar un elemento de la lista de producto en incrementos de funcionalidad.

- ***Scrum Master***: Es el encargado de que Scrum se entienda y se adopte correctamente dentro del *Scrum Team* . Es el líder del proyecto, pero que a su vez está al servicio del del *Scrum Team* ayudándoles a encontrar soluciones a los problemas encontrados así como es responsable de la eficiencia y del cumplimiento de los objetivos de este equipo.Además permite que personas externas a este equipo entienda que interacciones pueden ser útiles y cuales no. Estas interacciones son clave, pues el *Scrum Master* tiene el objetivo, y res responsable de maximizar el valor del producto

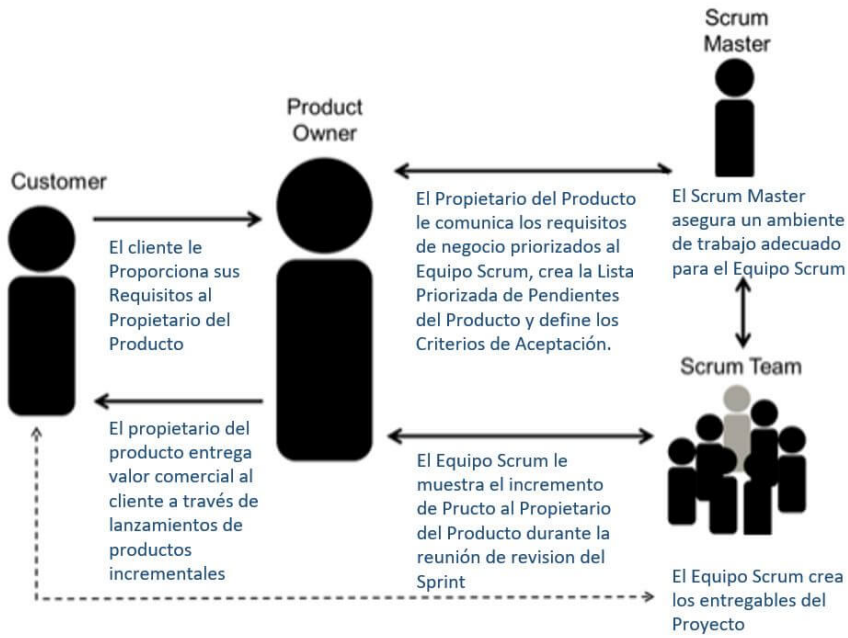

Figura 3.1: Roles e interacción. Obtenida de [60]

3.1.3. Eventos Scrum

En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (*time-boxes*), de tal modo que todos tienen una duración máxima. [34] (Podemos encontrarnos una visión de este proceso en la figura 3.2 Entre los eventos que nos podemos encontrar:

- ***Sprint***: Es el elemento base de Scrum, consiste en un bloque de tiempo de entre 2 semanas y 1 mes, donde se crea un incremento (Ver en la sección 3.1.4) que es marcado como 'terminado' en la finalización de este periodo. La duración de este bloque debe ser consistente a lo largo del proyecto, es decir, no debe variar la duración del sprint. Tras

la finalización de un sprint, inmediatamente comienza otro. Un Sprint podría cancelarse si el objetivo está desfasado respecto a una idea de mercado actual.

- ***Sprint Planning***: Previo al comienzo del Sprint se realiza una planificación de este, donde esta planificación es realizada por el equipo Scrum al completo, y tiene un máximo de duración de 8h para un sprint de un mes.
  Se trata por tanto de fijar los objetivos y tareas que se realizará en este sprint. A estas tareas se le asigna una estimación de carga, formada por puntos de historia, y que es representada en forma de horas en cuanto a trabajo equivalente.(Existen varias técnicas como la escala lineal o estimación de póker, que asignan un número predefinido a cada punto) De esta manera las tareas quedan representadas en horas de trabajo dentro de un sprint.

- ***Sprint Goal***: Es una meta establecida en el sprint en cuestión, y que se desglosa en Historias de Usuario (H.U) del Backlog (Ver sección 3.1.4 pertenecientes al sprint. De esta manera sirve de punto referencia mental a la hora de realizar de desarrollar la funcionalidad requerida.

- ***Daily Scrum***: En español *Scrum diario*, es una reunión diaria de unos 15 minutos donde el *Team Scrum* donde se sincroniza y evaluado el trabajo realizado en las últimas día adecuándolo al *Sprint goal*, y se planifica el de las futuras próximas 24h.

- ***Sprint Review***: En español *revisión del sprint*, es una renunión realizada al final del sprint con el objetivo de inspeccionar el incremento, modificar y adaptar la Lista de Producto (Ver sección 3.1.4). Se trata de una reunión informal con el objetivo de facilitar la retroalimentación de información y fomentar la colaboración.Participan Participan en ella *el Product Owner*, el *Development Team*, y *el Scrum Master*.

- ***Sprint Retrospective***:En español *retospectiva del sprint*, es una reunión que se realiza al final del sprint y tiene como objetivo inspeccionar el proceso realizado en el último sprint, identificando aquellos elementos que han salido satisfactoriamente, que pueden ser objeto de mejora en futuros sprints. De esta manera se logra la mejora en la forma de trabajo del *Scrum Team*. Tiene una duración máxima de 3h para un sprint de 1 mes.

- ***Sprint 0***: Corresponde a la fase previa al inicio de un proyecto de Scrum, donde se realiza un estudio previo del campo del proyecto, así como se prepara la arquitectura, una planificación inicial y una versión incial del Backlog (Ver sección 3.1.4) Cabe decir que este "sprint"no genera ningún valor al producto, y por tanto muchos miembros de la comunidad de Scrum rechazan esta nomenclatura, considerándolo simplemente una fase previa.

3.1.4. Artefactos Scrum

Los artefactos de Scrum tiene como objetivo representar el trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección

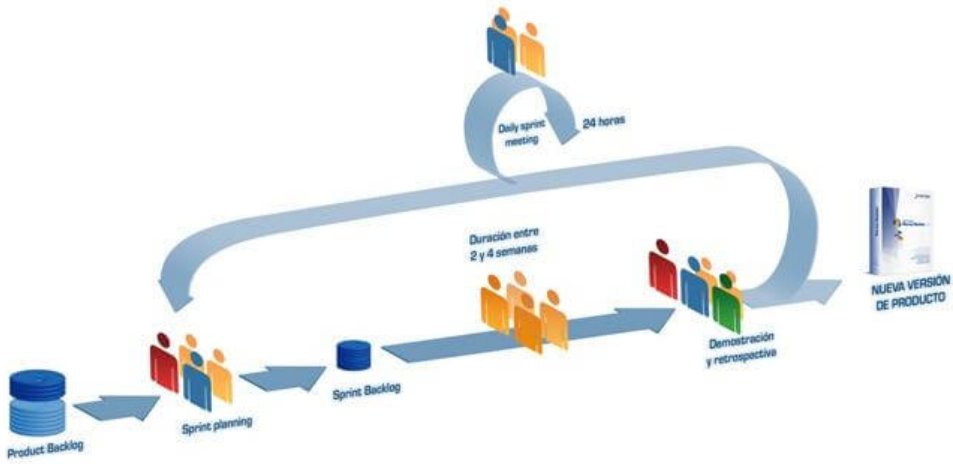

Figura 3.2: Eventos y artefactos. Obtenida de [54]

y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, necesaria para asegurar que todos tengan el mismo entendimiento del artefacto. [34] En la figura 3.2 podemos visualizar también los artefactos citados a continuación:

- ***Product Backlog***: En español *Lista de Producto* es una lista ordenada por prioridad de los requisitos del producto. Cabe decir que esta lista nunca está completa, se establece un *Backlog inicial*, que refleja los requisitos conocidos y mejor entendidos al principio.El *Product Backlog* sufrirá cambios a lo largo del proyecto, y por tanto es variable.Las personas encargadas de proporcionar las estimaciones en el *Product Backlog* son los integrantes del *Development Team.*El Dueño de Producto (*Product Owner*) es el responsable de la Lista de Producto, incluyendo su contenido, disponibilidad y ordenación. Los requisitos del producto son denominados historias de usuario.

- ***Sprint Backlog***, en español *Lista de Pendientes del Sprint*, es el conjunto de historias de usuario de la Lista del *Backlog* seleccionadas para el Sprint actual, sumado a un plan para entregar el Incremento de producto y conseguir el *Sprint Goal*. Una vez seleccionada esta lista, es inmutable mientras está en desarrollo, con el objetivo de optimizar el trabajo obteniendo así una visión de cuan cerca se está del *Sprint Goal.*

- ***Incremento***: Es la suma de todos los elementos del *Backlog* completados en el sprint actual, más aquellos que anteriormente ya habían sido completados en sprints anteriores. Al final de un Sprint el nuevo Incremento debe estar "Terminado", lo cual significa que está en condiciones de ser utilizado

3.2. Planificación

3.2.1. Adaptación de Scrum al proyecto

En el contexto explicado en la sección 3.1 con una variabilidad de los requisitos a lo largo del proyecto debido a la incertidumbre de las tecnologías escogidas es necesario aplicar la metodología de Scrum sobre el marco de este proyecto.

Primariamente estableceremos los roles, donde el alumno será *Product Owner*, así como será el único miembro activo en el *Development Team*. Finalmente el Rol de *Scrum Master* será ocupado por Joaquín Adiego, co-dirigente del TFG, así como Natalia Martín, co-dirigente también del TFG ocupará el rol de *stakeholder*.

Quedan establecidas las reuniones mediante videoconferencia las revisiones del sprint y retrospectivas, de la misma forma que en esta reunión se trata la planificación del siguiente sprint. En caso de indisponibilidad de alguna de las partes, por diversos motivos, ya sea por las prácticas del alumno u otro motivo, con motivo de no retrasar los sprints y cumplir los plazos la comunicación sería de forma escrita, pero con una explicación detalla e exhaustiva tratando así de evitar desviaciones en los objetivos planteados.

Existirá también una comunicación constante semanal vía correo con el *Scrum Master* con el objetivo de solventar problemas que requieran de su intervención.

Debido a la incertidumbre relativa al proyecto, se realizará un sprint 0 con el objetivo de analizar las tecnologías más apropiadas para la realización del proyecto. De esta manera este sprint servirá como preámbulo a la redacción del capítulo 2.

Se ha establecido para el valor de los puntos de historia la técnica de póker [35].El Planning póker es una técnica para estimar usando *story points* en donde cada miembro del *Development Team* debe otorgar un estimado en *story points* al ítem que se esté tratando, usando una sucesión de Fibonacci (0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89).

3.2.2. Planificación inicial de los sprints

El plan de estudios del grado en Ingeniería Informática establece que la duración del TFG son 300h. La fecha final del proyecto también será acorde a la fecha límite de entrega de la convocatoria ordinaria del TFG.

Por tanto, se ha establecido la duración de los sprints en 2 semanas, teniendo cada uno una carga de 34h aproximadamente por sprint. En la tabla 3.1 puede verse una planificación inicial. Esta planificación es susceptible a cambios, así como la fecha de finalización podría basarse en la fecha límite de la convocatoria extraordinaria.

Sprint	**Comienzo**	**Fin**
Sprint 1	15/02/2021	01/03/2021
Sprint 2	01/03/2021	15/03/2021
Sprint 3	15/03/2021	29/03/2021
Sprint 4	29/03/2021	12/04/2021
Sprint 5	12/04/2021	26/04/2021
Sprint 6	26/04/2021	10/05/2021
Sprint 7	10/05/2021	24/05/2021
Sprint 8	24/05/2021	07/06/2021
Sprint 9	07/06/2021	21/06/2021

Tabla 3.1: Planificación inicial de los sprints

3.2.3. Descripción de historias de usuario

Las historias de usuario (H.U) son abordadas en el capítulo 9 donde se describen las tareas asignadas, y si se han incorporado nuevas. Sin embargo existe una planificación inicial del *Product Backlog* que podrá variar en el tiempo. Se puede ver en la tabla 3.2 un *Backlog* inicial.

3.2.4. Modificaciones de la planificación inicial.

En esta sección vamos a hablar de los cambios ocurridos sobre la planificación inicial.

Debido a problemas técnicos con el equipo del alumno, en los sprints 6 y 7, donde no ha podido realizar sus labores se ha modificado la planificación en las fechas de los sprints. Por tanto se ha retomado el sprint 6 en la fecha original del sprint planificado inicialmente como el 8 (Día 24/05/2021). De esta manera se ha desplazado el calendario de sprints planificado inicialmente, manteniendo el número de sprints restante y con la misma duración de dos semanas. La finalización del proyecto por tanto se establecería sobre la fecha límite del proyecto.

Se ha añadido a las historias de usuario la funcionalidad necesaria para que haya notificaciones dentro de la aplicación. Puede verse en el capítulo 9 la variación que se ha producido en los diversos sprints.

En cuanto a la validación inicialmente se proponía la idea de que solo el grupo validase la tarea, aunque como veremos en la sección 5.3.1 finalmente también la persona que crea la tarea validará la entrega.

H.U	Nombre	Descripción
1	Consultar tareas	Como usuario quiero ver las tareas activas que tengo disponibles
2	Consultar historial	Como usuario quiero ver el historial de las tareas finalizadas
3	Consultar información de grupo	Como usuario quiero ver que miembros están formando actualmente mi grupo
4	Entrega	Como usuario quiero poder realizar una entrega para una tarea.
5	Validación de tarea	Como usuario quiero poder marcar una tarea como validada.
6	Lista de entregas	Como administrador quiero poder ver las entregas de una tarea
7	Registrar usuarios	Como administrador quiero poder limitar quien se registra en la aplicación
8	Añadir tareas	Como administrador quiero poder añadir nuevas tareas
9	Modificar tareas	Como administrador quiero poder modificar las tareas creadas
10	Crear grupos	Como administrador quiero poder crear grupos de trabajo
11	Modificar grupos	Como administrador quiero poder modificar los grupos creados
12	Crear proyecto	Como administrador quiero poder crear los proyectos creados
13	Eliminar grupos	Como administrador quiero poder eliminar los grupos de trabajo
14	Eliminar tareas	Como administrador quiero poder eliminar las tareas creadas
15	Consultar entrega	Como usuario quiero poder consultar el estado de una entrega
16	Login	Como usuario quiero poder *loguearme* para acceder a la aplicación
17	Cerrar sesión	Como usuario quiero poder cerrar la sesión actual.
18	Modificar proyectos	Como administrador quiero poder modificar los proyectos creados

Tabla 3.2: Descripción de las historias de usuario

Capítulo 4

Plan de riesgos y estimación de costes

Un riesgo es un evento o condición probable cuya ocurrencia tiene un efecto en los objetivos de un proyecto. [39] Por tanto este evento puede ocurrir en un futuro, y en caso de producirse tiene asociado un efecto. Se pueden categorizar estos riesgos en:

- Riesgos de proyecto: Son aquellos riesgos asociados relacionados con el no cumplimiento de los objetivos marcados por el equipo de desarrollo y el jefe de proyecto.
- Riesgos de negocio: Asociados al aspecto económico. Por ejemplo, el retraso del proyecto puede desenvocar que la tecnología ya no sea atractiva y las ventas no sean las esperadas.

Por tanto debe realizarse una estimación de los riesgos existentes, estimar el impacto de este, la probabilidad, y la importancia. Todo esto lo abordamos en el plan de riesgos que desarrollaremos a continuación.

Identificación de riesgos

Como dijimos en la sección 1.3, esta aplicación está destinada a ser usada en el ámbito escolar, si se estimase oportuno su uso. Por tanto, en un principio aunque tuviese un mercado secundario comercial, no está pensado en el desarrollo inicial de este proyecto su comercialización, y por tanto quedaría subordinado a otro proyecto que mejorase la implementación actual.

El resultado del análisis de riesgos se muestra en las tablas 4.1 a 4.8. En ellas se identifica el riesgo, estableciendo una probabilidad (Escala baja media alta), con un impacto en caso de producirse (Bajo, medio,alto). De la misma manera existirá un plan de mitigación (Reducir

la probabilidad de que ocurra) así como de contingencia (Plan de actuación en caso de que se materialice el riesgo)

Riesgo encontrado	Variación en las historias de usuario del proyecto
Probabilidad	Alta
Impacto	Medio
Plan de mitigación	Se ha seleccionado una metodología ágil con el fin de poder responder a los diversos cambios ocasionados en las historias de usuario por las restricciones que puedan existir en las tecnologías
Plan de contingencia	Modificación del backlog

Tabla 4.1: Riesgo 1: Variabilidad de historias de usuario.

Riesgo encontrado	Problemas de formación en las tecnologías del proyecto
Probabilidad	Media
Impacto	Alta
Plan de mitigación	Se han adquirido una serie de cursos de Udemy con el objetivo de mejorar la calidad de formación
Plan de contingencia	Buscar nuevas fuentes de formación. o buscar una tecnología alternativa.

Tabla 4.2: Riesgo 2: Problemas en la formación del alumno.

Riesgo encontrado	Falta de conocimiento para aplicar una arquitectura *blockchain*
Probabilidad	Baja
Impacto	Alta
Plan de mitigación	Se ha realizado un estudio de la arquitectura de aplicaciones similares
Plan de contingencia	Consultar a expertos, o foros de Ethereum para tratar de solventar los posibles problemas

Tabla 4.3: Riesgo 3: Falta de conocimiento para realizar el proyecto

Riesgo encontrado	Imposibilidad de realizar reuniones con el Scrum Master
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	Se ha establecido una posible franja horaria semanal.
Plan de contingencia	Comunicación por correo.

Tabla 4.4: Riesgo 4; Imposibilidad de realizar reuniones.

Riesgo encontrado	Retraso en el desarrollo de tareas
Probabilidad	Medio
Impacto	Medio
Plan de mitigación	Planificación no optimista en cuanto a duración de las tareas
Plan de contingencia	Nueva planificación de las tareas

Tabla 4.5: Riesgo 5: Retraso en el desarrollo de tareas

Riesgo encontrado	Baja temporal en el equipo de desarrollo (Salud, problemas técnicos..)
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Se puede contar con un sprint de refuerzo en la fase final.
Plan de contingencia	Modificación de la planificación inicial de los sprints, y mover las tareas a futuros sprints.

Tabla 4.6: Riesgo 6: Baja por enfermedad o problemas técnicos.

Riesgo encontrado	El proyecto no puede ser desplegado en redes públicas por seguridad.
Probabilidad	Bajo
Impacto	Alto
Plan de mitigación	Se ha realizado un estudio, y se puede implementar una conexión segura.
Plan de contingencia	El proyecto se limitaría a redes privadas de internet.

Tabla 4.7: Riesgo 7: Problemas de despliegue de la aplicación

Riesgo encontrado	El proyecto no cumple con la primera fecha límite
Probabilidad	Medio
Impacto	Medio
Plan de mitigación	Se ha previsto una segunda fecha límite implicando un sobrecoste del proyecto
Plan de contingencia	Modificación de la planificación y incorporación de nuevos sprints. Sobrecoste del proyecto

Tabla 4.8: Riesgo 8: Problemas en el cumplimiento de la ficha final inicial.

4.1. Estimación de costes

Para establecer el coste del proyecto, nos basaremos en la remuneración que debería recibir un programador informático en España por las horas realizadas para completar el proyecto. Por tanto el coste del proyecto será el dinero que se necesite para cubrir las horas de trabajo del programador, así como gastos extras que se requieran para formar al programar, y las herramientras software necesarias tanto para el desarrollo, como el hardware para el despliegue.

Por tanto, el proyecto tiene una duración establecida de 300h inicialmente, según la estimación de la Universidad de Valladolid, para el TFG en el grado en Ingeniería Informática.

Según el BOE [16], un programador recibe de media anualmente 22.892,29 euros (Bajo la condición: Personal con retribución mensual o diaria (extras aparte). Anualmente un trabajador realiza 1.792 horas. Por tanto, un informático cobra de media 12,77 euros.

En cuanto a las herramientas software, y hardware utilizado, y su repercusión en cuantro al coste dentro del proyecto:

- El despliegue se realizará sobre las máquinas virtuales de la universidad, por tanto, no existe un gasto extra en este aspecto.
- Firebase se utilizará en su plan gratuito. Las herramientas de desarrollo de software como Visual Studio o Astah no suponen costes extras, pues o son gratuitas, o se usa un plan gratuito. Se utilizará GitHub en su plan gratuito.
- Para posibles diseños se utilizará la página draw.io, que es gratuita.
- Gastos en formación de Udemy: Curso de desarrollo de react 19'99 euros, así como un curso de desarrollo de aplicaciones basadas en Ethereum, que tenía el mismo precio. Por tanto se estiman 40 euros en formación.

Se establece un posible sobrecoste de un 15 %, pues generalmente la precisión de la estimación inicial suele variar entre 15-20 %. [26]. Por tanto, la suma inicial asciende a 3.831 euros. Se estima por tanto con sobrecostes derivados de una posible desviación de la duración del proyecto u otros motivos un total de 4.405,65 euros.

4.2. Análisis final de costes.

Las horas estimadas fueron 300h, es decir partíamos de la idea impuesta dentro del plan de estudios. Finalmente el número final de horas asciende a 354h.

El número de sprints se ha mantenido, pero la fecha final del proyecto se ha modificado por problemas del alumno, expuestos en la sección 3.2.4

El coste por 354h asciende a 4521 euros. Por tanto el sobrecoste estimado era cercana a esta posibleoscilación existente en las horas del proyecto. Finalmente el sobrecoste asciende a 690 euros, y no los 574 previstos inicalmente.

Capítulo 5

Análisis previo

Se ha realizado un análisis previo para tener un mejor enfoque del proyecto, estando esta tarea enmarcada en el sprint 0 de Srum. Cabe decir que este concepto puede variar a lo largo de los diversos sprints, y que por tanto los diagramas que se ven a continuación son minimalistas y no representan la implementación final.

5.1. Modelo de dominio conceptual

En la figura 5.1 podemos ver el modelo de dominio conceptual. Este ha sido realizado en base a las historias de usuario recogidas en el *Backlog* de la sección 3.2.3. Como hemos dicho, es susceptible a cambios, y representa la idea inicial tomada.

5.2. Funcionamiento de una DApp

Las aplicaciones descentralizadas que se ejecutan sobre la *blockchain*, basan su funcionamiento en el uso de esta para el registro de transacciones, así como en el uso de *smart contracts*. La idea más purista basa su idea general en la descentralización de todos los servicios, otras mas híbridas sin embargo sacrifican la descentralización con el objetivo de ganar funcionalidad. En la figura 5.2 podemos ver una visión general de un modelo centralizado y uno descentralizado.

Por tanto una aplicación basada en *blockchain* utiliza esta como base de datos descentralizada. Sin embargo, existe una serie de restricciones a la hora de almacenar información en esta:

- No se debe guardar información confidencial como contraseñas.

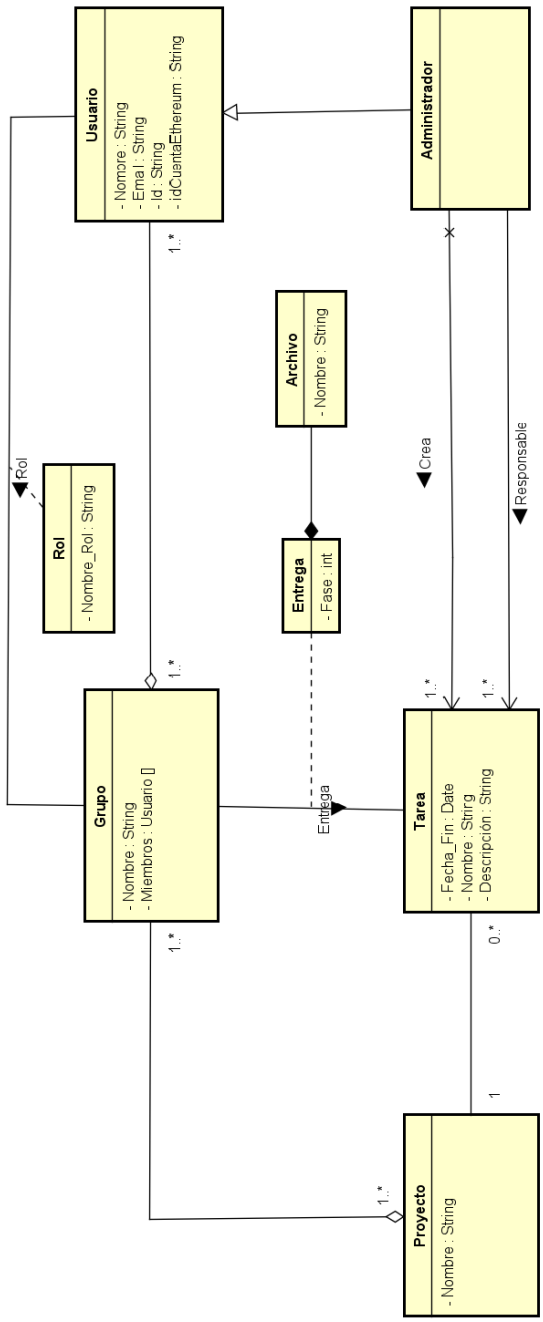

Figura 5.1: Modelo conceptual de dominio.

- No se debe almacenar información que viole la normativa actual de privacidad.
- Computacionalmente el almacenamiento de archivos es inasumible, y por tanto se deben de buscar otras opciones.

Para las dos primeras problemas se suele usar un sistema de bases de datos descentralizado, o bien distribuido. (Recordemos que en una red distribuida siempre pasa por una serie de nodos centrales, mientras que en una red descentralizada esto no ocurre. Para más información consultar [17] [45])

En el caso de esta aplicación se utilizará la segunda opción, en el capítulo 6 abordaremos la tecnología seleccionada.

En el caso del tercer es necesario un sistema distribuido de almacenamiento basado en direcciones hash. De esta manera no se almacenará en la *blockchain* el archivo, si no que se ejecutará un contracto donde queda almacenada la dirección hash del documento. De esta manera este archivo es también eliminable y cumple con la normativa de privacidad en caso de ser información que deba ser borrada. En el capítulo 6 abordaremos la tecnología seleccionada.

Finalmente puesto que la aplicación está orientada a redes privadas en su mayoría, se ha seleccionado la opción de mantener un servidor web centralizado como solución, de esta manera actuará también como proveedor, y no utilizaremos Metamask como vimos en la sección 2.3.1. Básicamente el servidor alojará un nodo Ethereum con todas las cuentas, y serán utilizadas en remoto vía este proveedor. En la sección 6.1 se abordará con mayor detalle el funcionamiento de esta tecnología.

Finalmente podemos hacernos una idea de como estará estructurada la aplicación, en la figura 5.3 podemos ver un esquema conceptual de la arquitectura del sistema.

5.3. Análisis de la funcionalidad necesaria

5.3.1. Sistema de validación de tareas

La idea base es que un proyecto esté formado por una serie de grupos de usuarios. A estos grupos se les asignarán una serie de tareas. Cada grupo por tanto antes de la fecha límite debe realizar una entrega para esta tarea y validarla. El sistema de consenso de validación de la entrega antes de insertar la entrega en la *blockchain* es el siguiente, dividido en 4 fases:

1. El grupo tendrá un responsable (Responsable de grupo) que será el encargado de realizar la entrega. Pasará a fase 2 la tarea.
2. Cada uno de los miembros dará el 'sí' explicito a la entrega, es decir la valida. Una vez todos validan la tarea pasa a fase 3.

3. La persona que creó la tarea (Responsable de la tarea) deberá validar la tarea también. Pasará a fase 4.

4. El responsable deberá por tanto marcar la tarea como finalizada, y se producirá la inserción del documento en la *blockchain*.

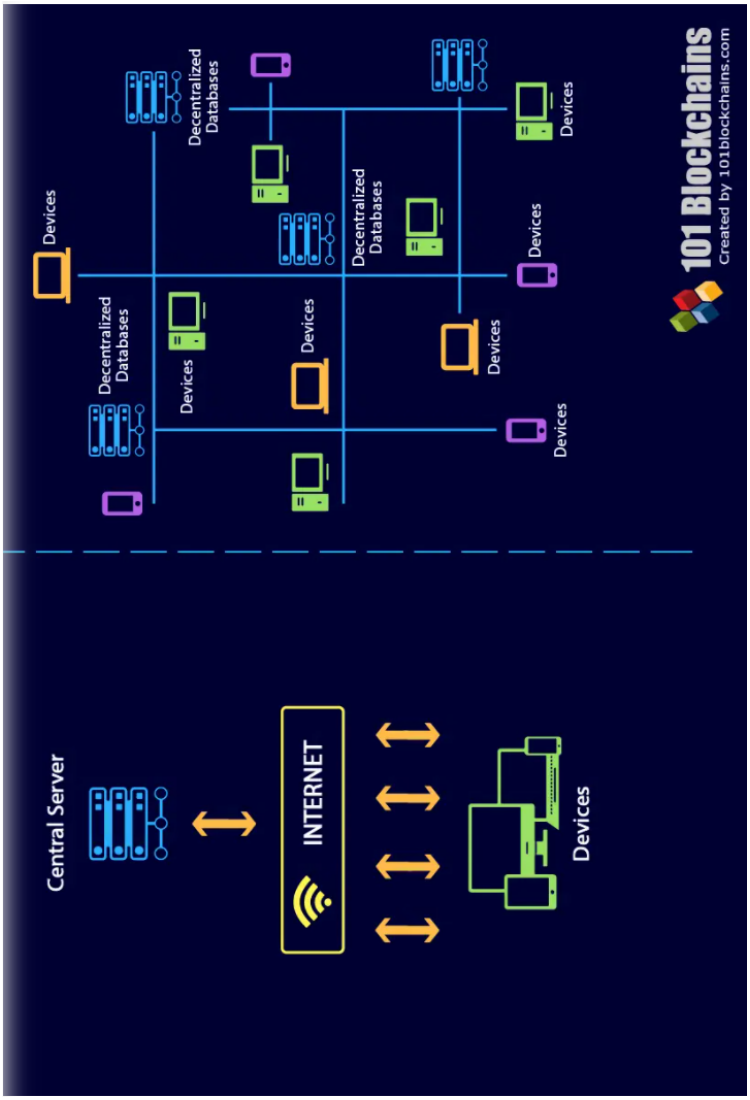

Figura 5.2: Descentralizado vs centralizado. Obtenida de [18]

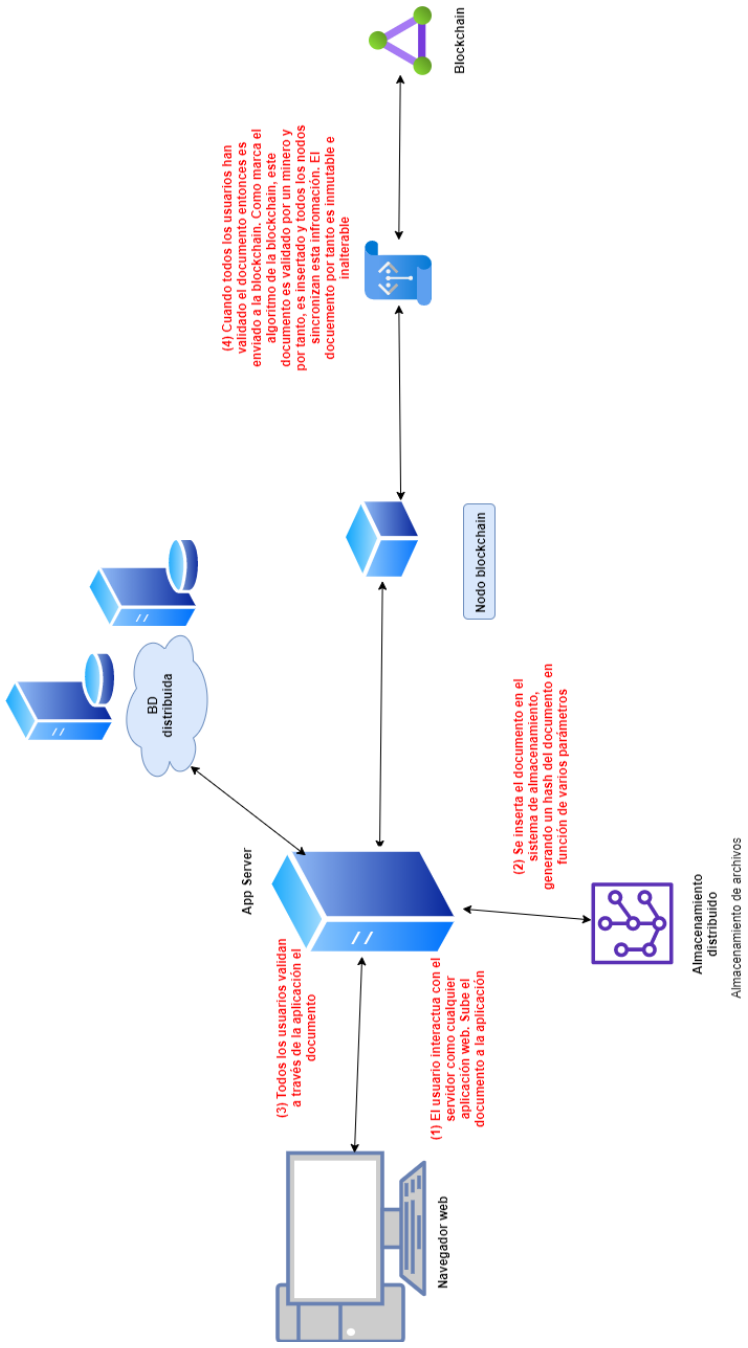

Figura 5.3: Esquema conceptual de la arquitectura.

Capítulo 6

Tecnologías utilizadas

6.1. Ethereum

Ethereum es una plataforma open source, que sirve para programar contratos inteligentes. Sus aplicaciones web son denominadas como la web '3.0', debido a la descentralización de los servicios

Se ha seleccionado esta tecnología al ser la que más está creciendo actualmente, contando con una amplia comunidad de desarrolladores y con una documentación en crecimiento y estable.

En Ethereum hay dos formas de gestionar las cuentas, de forma local o remota. La primera forma se basa en que cada usuario genera una dirección privada (Clave privada), que es gestionada por él, y que es utilizada para firmar las transacciones que enviará a la *blockchain*. Permite garantizar el anonimato de la persona que posee esta cuenta, pues es necesario informar de la dirección pública (clave pública) para que se comuniquen contigo.

La segunda de estas formas, es la utilizada en este proyecto. Todas las claves privadas están alojadas en un único nodo, en este caso en el servidor. Esta decisión inicial de diseño, ha sido para facilitar la accesibilidad de los usuarios, impidiendo así que tengan que desplegar un nodo, o utilizar una extensión como Metamask, evitando así que instalen software de terceros, y sea accesible desde cualquier dispositivo. Al ser creadas de forma remota, estas cuentas necesitan de una contraseña o frase con la que se cifra la cuenta. Cuando sea necesario realizar una transacción, la cuenta debe ser desbloqueada con esta contraseña.

Esta red será privada, y por tanto estará limitado el acceso y creación de cuentas. De esta manera, también será ficticio el balance de las cuentas, y el computo necesario será mucho menor.

6.1.1. Web3JS

Web3JS (El 3 viene de la web '3.0' denominada por ellos y JS de JavaScript) es una API en Javascript compatible con Ethereum que implementa las especificaciones genericas RPC en formato JSON. Esta librería se comunica con un nodo local a través de llamadas RPC (Sustituido en nuevas versiones por HTTP), y es compatible con los diversos tipos de nodos Ethereum que existen. [67]

Esta tecnología por tanto es la utilizada como proveedor, utilizando estos mecanismos de RPC (Y ahora HTTP en nuevas versiones) para interactuar desde el nodo con la web.

6.1.2. Clientes de Ethereum

Existen varias implementaciones de Ethereum, en diversos lenguajes. En mi caso se ha seleccionado GoEth, que está escrita en Go. Basicamente estas implementaciones son ejecutables via CLI. (Consola) [32]

6.2. React

React es una librería Javascript focalizada en el desarrollo de interfaces de usuario desarrollada por Facebook, y enfocada a páginas de tipo *Single Page Tipe (SPA)*. La propia página web se define así, y no como un "framework", como muchos tienden a pensar.

Básicamente React dentro de una arquitectura MVC estaría enfocado a la vista. Su objetivo por tanto es ayudar al desarrollador web, evitando así que tenga que desarrollar una gran cantidad de código javascript para lograr la funcionalidad.

6.2.1. Estructura de React

Como hemos dicho anteriormente React es una librería, y como librería que es, se implementa sobre archivos de tipo *JavaScript*. Existe una curiosidad y es que el lenguaje de estos archivos no es propiamente JavaScript, si no una extensión de este (JSX).

JSX es una extensión de sintaxis de JavaScript que nos permite mezclar JS y HTML (XML), de ahí su nombre JavaScript XML. Básicamente nos permite usar código JS dentro del HTML, y viceversa.

Al ser una librería y no un framework, existe una versatilidad en el uso de React. En el caso de este proyecto, se ha utilizado la herramienta *create-app* [22]. Esta herramienta permite establecer un proyecto basado en componentes, partiendo del componente padre App, y de un punto de inicio, que es el fichero index.html, que enlaza con dicho componente.

6.2.2. Ciclo de vida de un componente en React

Como hemos dicho anteriormente, el desarrollo de esta aplicación esta basado en componentes. Hasta la versión 16.8 de React, el desarrollo estaba basado en clases (*React class component*) donde el ciclo de vida venía marcado por una serie de funciones. Con la llegada de esta versión, se incorporaron los hooks de estado, que permitían un desarrollo más rápido de la aplicación, proporcionando más versatilidad y reducción del código de la implementación.

Para entrar en el mundo de estos hooks y del ciclo de vida, primero debemos presentar lo que es un estado en React. Un estado en React es, entonces, un almacén de datos mutable, asociado a un componente en concreto. Este estado es inicializado y puede ser modificado, y permite el acceso a sus valores desde ese componente.

Además del estado, los componentes tienen unas propiedades. Estas propiedades básicamente son una serie de variables o funciones que se reciben cuando se invoca al componente, y pueden ser utilizadas con diversas finalidades. Por ejemplo, un componente que reciba el título para luego que sea mostrado en vía HTML.

En este proyecto se ha utilizado los *function hooks* [58], la alternativa a las clases de componentes. Básicamente el funcionamiento es el mismo, pero requiere mucho menos código a la hora de realizar una implementación. En la figura 6.1 podemos ver un diagrama de actividad del ciclo de vida de un componente tradicional de React. (*Class component)*

- Cuando es llamado y comienza su ciclo de vida, y recibe las propiedades, ya sean funciones o variables.
- A continuación comienza el montaje el componente(*ComponentWillMount*). Este apartado es utilizado para establecer valores iniciales a los estados del componente, y inicializarlos por tanto.
- Tras este punto se ejecuta el render. El render contiene el código JSX que se mostrará al usuario. Puede tomar valores del punto anterior, por ejemplo, de una carga de la base de datos, para ser mostrados posteriormente estos datos en una lista al usuario.
- Tras este paso finaliza el montaje del componente, a este estado se le llama *componentDidMount.*Básicamente se suele utilizar para recibir datos de las APIs. Solo se ejecuta en el lado del cliente.
- Una vez finalizada esta carga todo dependerá de los eventos que se produzcan dentro del componente. Hay dos opciones, una actualización de estado del componente (De un estado de este básicamente) o la finalización del ciclo de vida de este componente(*ComponentWillUnmount)*, siendo destruido. Las actualizaciones se producen por eventos tales como un click sobre un botón, que actualiza un estado, entonces se produce la recarga del componente, y se produce una nueva llamada al render.

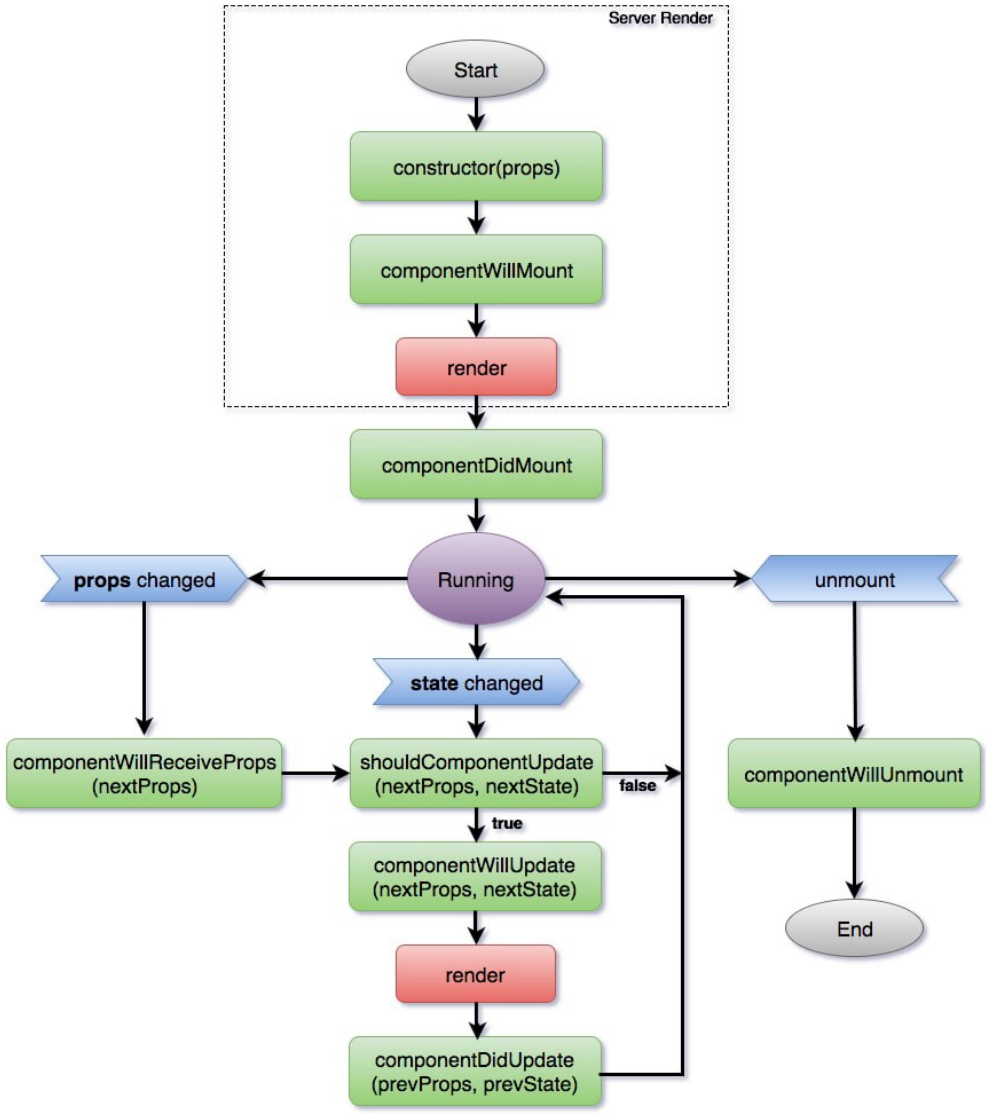

Figura 6.1: Ciclo de vida, componente React. Obtenida de [38]

Ciclo de vida en hook functions

Una vez explicado el funcionamiento en un componente de clase de React, veamos este nuevo mecanismo de hooks. Un hook permite la creación de un componente dentro de una función. Básicamente antes debíamos crear una clase y definir una serie de métodos para poder acceder a dicha librería, y poder utilizarla dentro de las funciones. Ahora simplemente podemos definir una función e invocar código React dentro de ella sin la necesidad de realizar el anterior proceso, que era bastante largo y tedioso. Existen una nueva serie de métodos opcionales dentro de una *function hook* [58]:

- useState: Permite la inicialización y definición de estados. [66]
- useEffect: [65] Sustituye al viejo ciclo de vida. Ejecuta los tres métodos: *ComponentWillMount*, *ComponentDidMount* y el *ComponentWillUnmount*. Basicamente el código que está dentro de este se ejecuta por primera vez haciendo la función de *ComponentWillMount*, permite establecer observables sobre los estados sustituyendo al *ComponentDidMount* y a todo el proceso de observación de estados anterior, y se puede establecer un return a esta función para ejecutar la funcionalidad del *ComponentWillUnmount*

Podemos ver el nuevo ciclo de vida en la figura 6.2. Los cambios más significativos son que ahora se ejecuta antes el render que el estado de ComponentWillMount, actualizandose por tanto posteriormente el componente con los valores cargados mediante el useEffect, o mediante la inicialización del useState.

6.3. IPFS

El Sistema de archivos interplanetario (En inglés *InterPlanetary File System - IPFS*) es una red (Y protocolo) diseñado para crear un método p2p direccionable con el objetivo de almacenar y compartir archivos, basado en un FS (*File system* ò sistema de ficheros) descentralizado

Básicamente IPFS utiliza el algoritmo SHA-256 [36] para transformar la ruta que indexa el documento. El resultado es un hash que recibe el nombre de *content identifier* (CID). Este CID es utilizado para obtener posteriormente el documento [37]

6.3.1. IPFS-http-client

Es una API en JavaSCript que implementa el *core* del protocolo IPFS. [42]

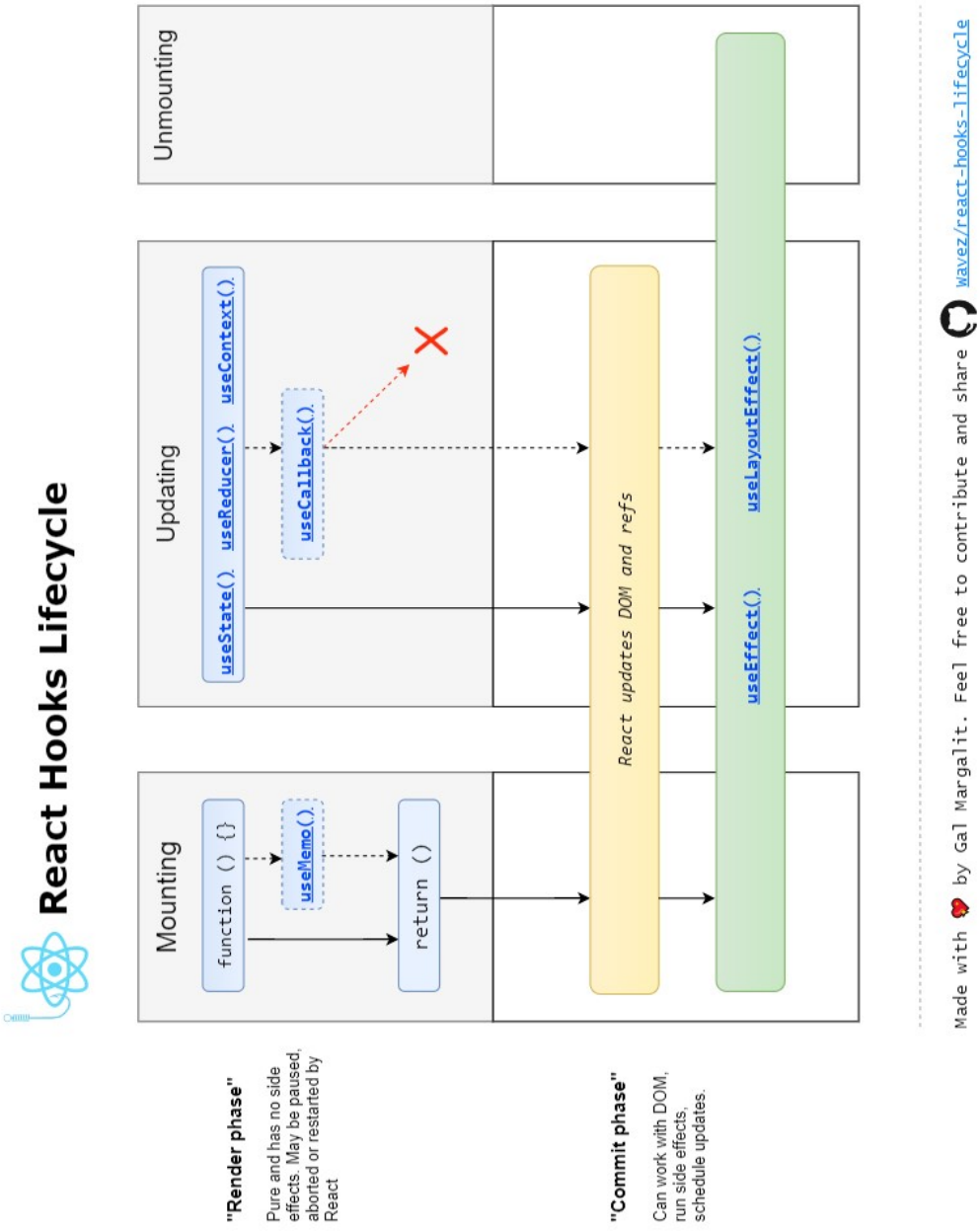

Figura 6.2: Ciclo de vida, hooks. Obtenida de [44]

6.4. NPM

NPM (Node Package Manager) es el administrador de paquetes predeterminado para el tiempo de ejecución de JavaScript Node.js, que permite añadir y gestionar dependencias, librerías y módulos a las aplicaciones que estamos desarrollando. Requiriendo NodeJS, permite desplegar un servidor en local para desarrollar la aplicación, así como una compilación para el despliegue. Entre los paquetes instalados destacan:

6.4.1. @material-ui/core

Este plugin contiene una librería de componentes listos para el uso, como tablas con funcionalidades de paginación, o selectores con búsqueda incluida. También contiene un listado de iconos y otras funcionalidades. [5]

6.4.2. React-bootstrap

Este plugin es la implementación de Boostrap para react. Proporciona una gran variedad de clases CSS, que permiten el desarrollo del front-end con facilidad, así como una serie de componentes como botones, inputs de formularios.. [57]

6.4.3. React-router-dom

React no tiene un sistema de navegación por defecto al tratarse de una librería. Por tanto este paquete nos proporciona una serie de componentes de navegación. Es por tanto el paquete encargado de las rutas, y de la navegabilidad entre ellas. [56]

6.4.4. Moment y moment-timezone

Paquete para el formateo de fechas, y gestión de zonas horarias de la aplicación. [46]

6.5. Solidity y Truffle.

Solidity es un lenguaje de alto nivel orientado a *smart contracts*. Su sintaxis es similar a la de JavaScript, de tipo estático y está enfocado específicamente a la Máquina Virtual de Etehreum (EVM). [62]

Truffle es un conjunto de herramientas de programación orientadas a *smart contracts* (contratos inteligentes) que permiten el desarrollo aplicaciones sobre la *blockchain*. El objetivo principal de Truffle es proveer un entorno de desarrollo en la *blockchain*, permitiendo con estas

herramientas la compilación de s*mart contracts*, la automatización del testeo de contratos y despliegues de contractos, y gestión de redes. [43] [15]

6.6. Firebase

Firebase es una plataforma integrada en Google Cloud para el desarrollo de aplicaciones web y móvil. Facilita la gestión de los usuarios, proporciona servicios de base de datos en tiempo real y de hosting, así como tiene un sistema de funciones en la nube, que permite la ejecución de estas en función de eventos.

En este proyecto se utilizará como base de datos (Real time firestore)

6.7. Jira

Jira es un software que permite planificar proyectos que utilicen metodologías ágiles como por ejemplo Scrum que utilizada en este proyecto. Jira permite planificar flujos de trabajo, pudiendo a partir de historias de usuarios crear un conjunto de tareas que son asignadas a los miembros, así como recoger los problemas que haya habido en ellas (Como el incumplimiento, denominado incidencia), entre muchas de las funcionalidades que tiene. [10]

La herramienta me ha permitido desde el listado del Backlog (Donde estaban las historias de usuario que recogí en la sección 3.2.3 y otras que se irán incorporando a lo largo del proyecto así como otras que desaparecen) asignar estas historias de usuario a los sprints, donde cada historia era transformada en una serie de tareas para lograr la funcionalidad objetivo.

6.8. Github

GitHub [30] un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso. [29]

Un control de versiones, es un sistema software que permite rastrear y gestionar los cambios realizados en uno o varios archivos que están establecidos en un repositorio(Lugar donde se alojan los archivos). El sistema de control de versiones permite por tanto tener un registro de los cambios realizados sobre estos archivos, permitiendo así revertir los cambios si fuese necesario, a la vez que permite analizar estos cambios para ser fusionados en caso de encontrar dos cambios a la vez sobre un mismo archivo.

Los repositorios por tanto pueden ser compartidos, y como hemos dicho, permiten a los desarrolladores trabajar simultáneamente gracias a la posibilidad de unificar los cambios realizados.

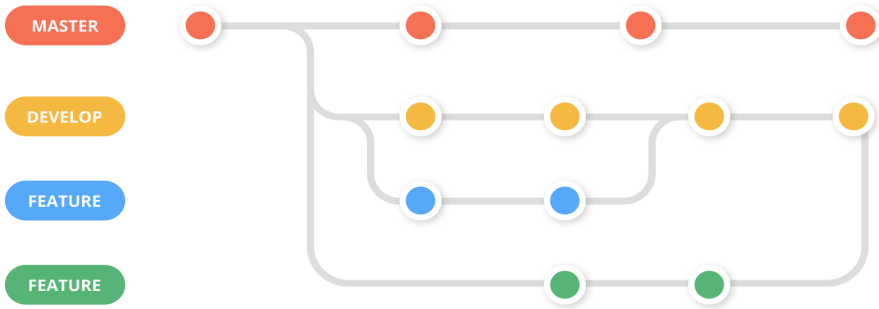

Figura 6.3: Ramas de Git. Obtenida de [6]

Git es el sistema de control de versiones distribuido más famoso. Se trata de un proyecto de código abierto, y por tanto se permiten realizar modificaciones en su código fuente. Su principal virtud es la ramificación, que permite trabajar a los desarrolladores dividir en ramas el trabajo, para posteriormente unificarlo.

En la figura 6.3 podemos ver un ejemplo de la ramificación de un proyecto. Generalmente los proyectos están establecidos en una rama master y una rama develop. La rama master solo obtiene los cambios de *develop*, es decir son unificadas (merge), cuando la funcionalidad está finalizada y testeada. La rama *develop* por tanto se subdivide en una serie de ramas por funcionalidad. En el ejemplo que vimos anteriormente, de esta rama colgaban dos, una para cada funcionalidad, que posteriormente eran unificadas con la rama develop.

6.9. Astah

Astah es una herramienta de modelado UML creada por la empresa Change Vision. Permite la creación de una gran variedad de diagramas, así como tiene funcionalidades como la generación de código en su versión profesional, o herramientas para la especificación de requisitos.

Se ha utilizado en este proyecto para el desarrollo del diagrama conceptual de dominio, así como para los diagramas de despliegue o de componentes. [52]

6.10. Draw.io

Draw.io es una página que permite realizar todo tipo de diseños, y de diagramas. He aprovechado por tanto las plantillas presentes en esta página para realizar los bocetos de la página web. [28]

Además esta herramienta ha sido utilizada para realizar el concepto de arquitectura que se vió en la figura 5.3

Capítulo 7

Diseño

Como dijimos en la sección 6.2, React ofrece una libertad a la hora de seleccionar una arquitectura. En el presente proyecto, se ha utilizado el patrón arquitectónico MVVM, aprovechando los nuevos *hooks* de estado, desarrollando a su vez un diseño basado en componentes. Así como se han utilizado otros patrones para los servicios de *back-end*.

7.1. MVVM

Modelo-Vista-Modelo de vista ò Model-View-ViewModel en inglés, es un patrón de diseño arquitectónico que permite la separación de a lógica de negocio, de la interfaz de usuario. Esta formado por tres partes: [59]

- Modelo (Model): Consta de la lógica de negocio formada por las diversas estructuras de datos.
- Vista (Vista): La Vista define cómo la información y las funcionalidades se mostrarán gráficamente, es decir, aquello que el usuario verá. En otras palabras es la interfaz de usuario.
- Modelo de vista (ViewModel): Enlaza la vista con el modelo. contiene los métodos para interactuar con el modelo.

Este patrón arquitectónico es bastante utilizado en el desarrollo de aplicaciones móviles, así como en frameworks SPA como Vue JS. Su principal caracteristica que permite diferenciarlo de otros patrones arquitectónicos, como MVC es la ausencia de un controlador real. Existe sin embargo un "binder"que permite realizar cambios sobre el modelo de forma automática, pasando los cambios a ser

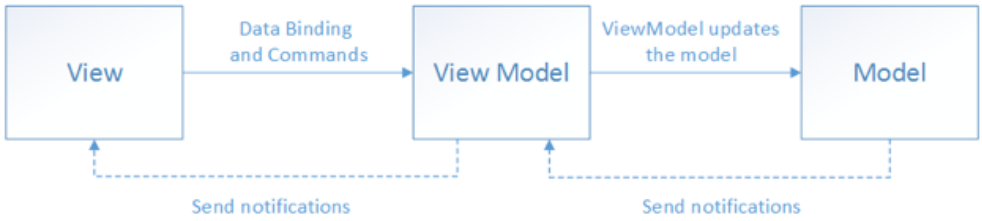

Figura 7.1: Ramas de Git. Obtenida de [23]

7.1.1. MVVM en el proyecto

En la sección 6.2 que React básicamente se ocupaba de la vista. Con la introducción de los nuevos *hooks* y *function hooks* que explicamos, esta funcionalidad ha permitido una mayor comodidad a la hora adoptar ciertos patrones (Generalmente Redux era uno de los más utilizados, aunque cada vez se está volviendo más obsoleto).

Hasta hace no mucho, la función de observadores en React era cubierta por la librería RxJS (Reactive Extensions Library for JavaScript), así como su implementación para Redux, una arquitectura muy utilizada en React. Pero con la llegada como decimos de los nuevos *hooks* de estado en React, y en concreto del *useEffect* [65] , esta librería ha sido reemplazada por este *hook*.

Análogamente, también se ha sustituido otra función que realizaba Redux, que era establecer 'un estado único' para a la aplicación. Es decir, cada componente tendría sus estados propios, y además, existiría un estado común para toda la aplicación. Esto actualmente se logra mediante el *hook useContext* [58], evitando así el problema que existía con el paso de propiedades a niveles más bajos de componentes hijos. También gracias a estos hooks, se ha sustituido el *reducer* [58] (Que básicamente permite tomar un estado anterior para generar otro. Básicamente se utiliza cuando un estado depende de otro anterior. De esta manera, no se realiza una recarga del componente).

Por último decir, que hasta hace poco, sin estos *hooks* de estados, existía la opción de aplicar la arquitectura MVVM de diversas, maneras. Una de ellas utilizando la aproximación React-MVx. [13]

Ahora sí, explicada esta terminología, la idea base para poder aplicar el MVVM:

- Como habíamos dicho hasta hora, React se encargaba de la parte de la vista, y así seguirá siendo.
- El *binder* y el *ViewModel* son ocupados por los *hooks* de estado. Es decir, al definir un estado, le indicamos a *useEffect* como parámetro que debe observar los cambios sobre dicho estado, de esta manera estamos logrando que cuando usamos este objeto de estado en la vista, la modificación ¡de la vista por parte del *ViewModel* sea automática.

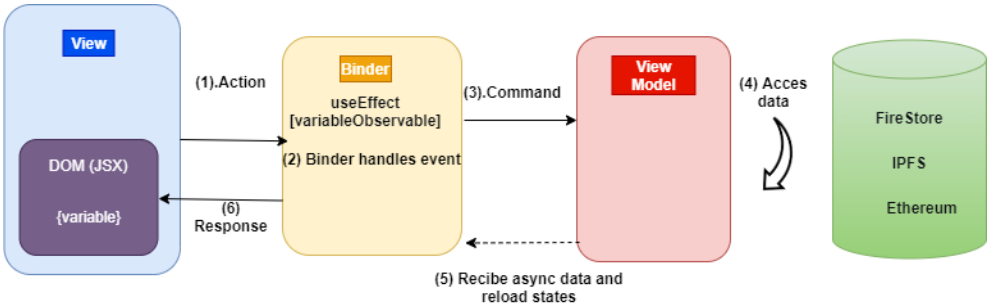

Figura 7.2: Esquema básico MVVM React realizado por mí.

- El modelo por tanto estará formado por el conjunto de datos que forman los estados de la aplicación

En la figura 7.2 he realizado une esquema básico de la aplicación de este patrón arquitectónico.

7.2. Off-chain Datasore Description

Este patrón de diseño[24], se basa su funcionamiento en la utilización de *hashes* a la hora de pasar una serie de datos que ocupan mucho en memoria secundaria (Almacenamiento). Esto ocurre en este proyecto a la hora de almacenar los documentos, en vez de almacenar este, queda un registro de la transacción con el id *hash* de este documento, y un id correspondiente a esa entrega. De esta manera tenemos un registro que indica que se realizó esa entrega.

7.2.1. Utilización de IPFS junto a la Ethereum

Como vimos en la sección 6.3 IPFS es utilizado para almacenar archivos, indexando en acceso a estos mediante un *hash*. En las figuras 7.3 y 7.4 podemos ver un esquema genérico del comportamiento.

- Una parte de la aplicación, en este caso al ser genérico se ha denominado como controlador, es el encargado de ejecutar las operaciones para insertar el documento o archivo en el sistema IPFS. Este documento es transformado previamente a formato *blob*. Posteriormente se ejecuta el *smart contract* que almacena la entrega y el documento.
- En el segundo diagrama, vemos el funcionamiento inverso, que básicamente obtiene el *smart contract* que ejecutó, obtiene el *hash*, y posteiormente con ese *hash* puede obtener a través del *gateway* el archivo.

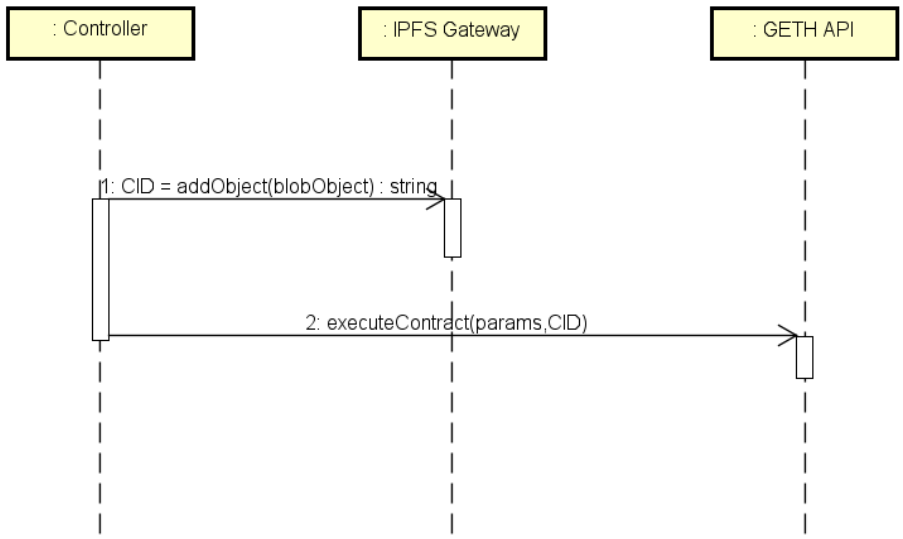

Figura 7.3: Comportamiento base para añadir un hash utilizando el patrón off-chain

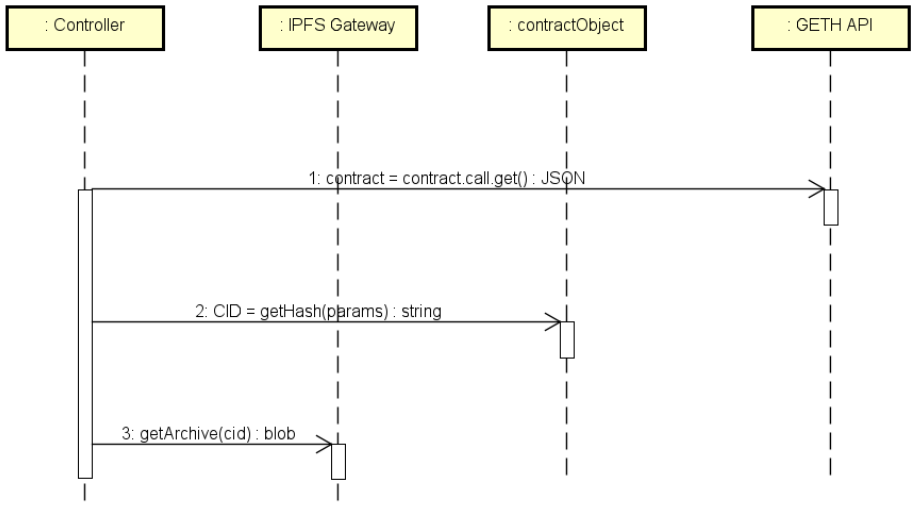

Figura 7.4: Comportamiento base para obtener un documento utilizando el hash

7.3. Atomic Web Desing

El concepto de Atomic Web Design nace de un artículo de Brad Frost [11], en el que explica el diseño de una interfaz web descomponiendo en unidades más pequeñas, como él denomina átomos, y reutilizando estos componentes a través de la composición para generar elementos más grandes y potentes que compongan todo el diseño. [64]

Este patrón por tanto permite la creación de componentes independientes que favorecen la reutilización, y facilitad el mantenimiento de estos al tratarse como piezas aisladas.

Cada uno de estos componentes en este proyecto estará en un archivo js. Se combina con el patrón de React *Conditional Rendering*, que básicamente subdivide a este componente en módulos dentro de ese fichero, y son llamados de forma dinámica en el render, en función de las condiciones que se establezcan.

7.3.1. Componentes de este proyecto

A continuación se mostrará un listado de los componentes desarrollados.

- **App** Componente padre del que cuelgan el resto de componentes. Contiene la aplicación entera. Además contiene el *Router* padre que permite la navegación a lo largo de la web.(Un router en react es un componente que permite el enrutamiento entre enlaces.)
- **PanelAdminTask** Contiene el router hijo para el enrutamiento dentro del panel de tareas del admin.Es responsable por tanto de invocar al menú secundario que permite la navegabilidad entre sus componentes hijos.
- **TaskListaAdmin**. Contiene la lista de tareas activas y de entregas.
- **AddTask**. Es el componente que permite a un administrador crear tareas.
- **ValidateTask**. Es el componente que permite a un administrador, que ha creado una tarea, validar las entregas asociadas a esa tarea.
- **ManageTask** Componente para la modificación de las tareas (Grupos asociados ect..)
- **AddProject** Componente para añadir tareas
- **ManageProject** Componente que contiene la lista de proyectos y que permite la edición de estos.
- **PanelAdminGroupAndUser**. Contiene el router hijo para el enrutamiento dentro del panel admin de grupos y usuarios. Además invoca a un segundo menú.
- **RegisterUser** Componente que permite registar usuarios en el sistema.
- **ManageUsers** Componente que permite gestionar usuarios.

- **ManageGroup** Componente que permite la gestión de grupos, y contiene un listado de ellos.
- **AddGroup** Componente que permite crear grupos.
- **Profile** Componente que muestra el perfil de usuario.
- **MyGroups** Componente que muestra los grupos de un usuario.
- **Notificaciones** Componente que muestra las notificaciones de usuario.
- **Login** Componente que permite el acceso al sistema.
- **HeaderBar** Barra principal de navegación superior.
- **SecondaryHeaderBar** Barra secundaria invocada por otros componentes como PanelAdminGroup, para garantizar la navegabilidad en sus componentes hijos
- **List Task** Contiene la lista de tareas disponibles de un usuario.
- **Task** Permite realizar una entrega, validar, y consultar el estado de esta.

En la figura 7.5 se puede ver los anteriores componentes citados, así como la relación que existe entre ellos.

Existe además un componente denominado PrivateRoute, que ha sido creado para gestionar las rutas en función de si el usuario está logueado o no.

7.4. Backend as Service

Un *BaaS* o *mBaaS* o *Backend as a Service* (Backend como servicio) es una plataforma que facilita el desarrollo del lado del *back-end* proporcionando al desarrollador una serie de métodos para interactuar con los servicios alojados en él. [12]

Por tanto las responsabilidades de ejecución y mantenimiento de servidores caen a cargo de un tercero, permitiendo al desarrollador centrarse en el desarrollo del *front-end*, haciendo uso se los servicios proporcionados. Podemos ver una comparativa del esfuerzo de desarrollo del *front-end* frente al *back-end* en la figura 7.6

Se ha seleccionado en este proyecto Firebase, debido a la cantidad de servicios que proporciona como por ejemplo una base de datos en tiempo real, o la autentificación y validación de usuarios. De esta manera, descentralizamos del servidor donde está alojado el nodo Ethereum, los servicios de base de datos.

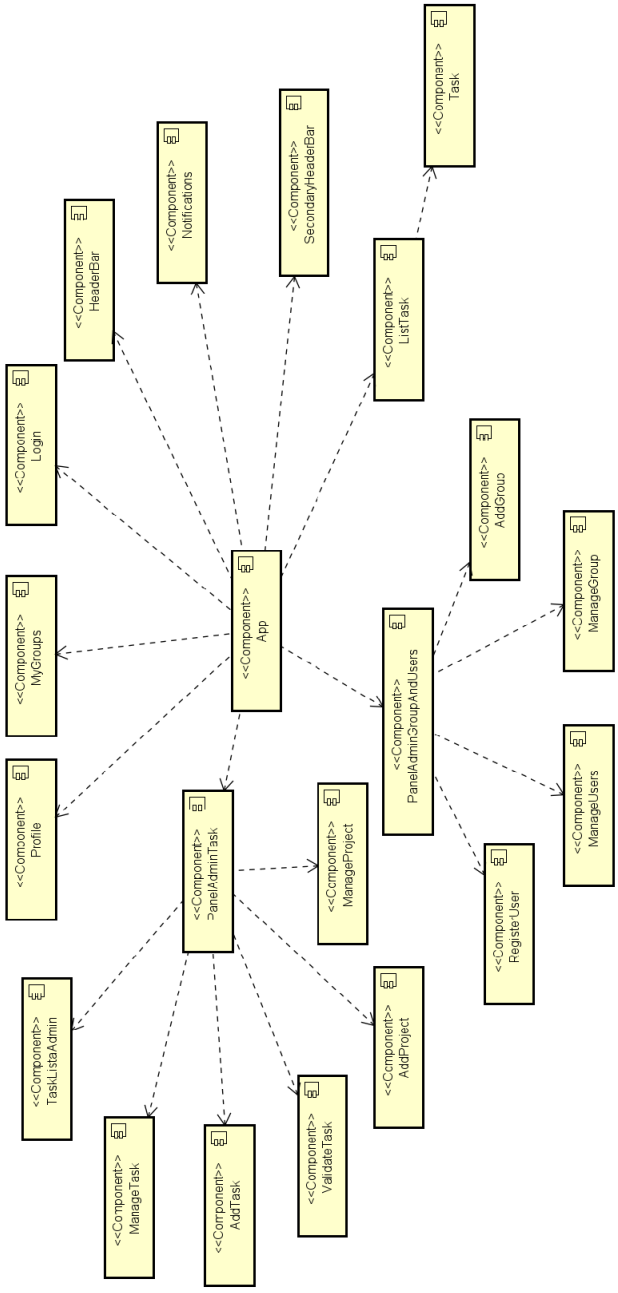

Figura 7.5: Diagrama de componentes.

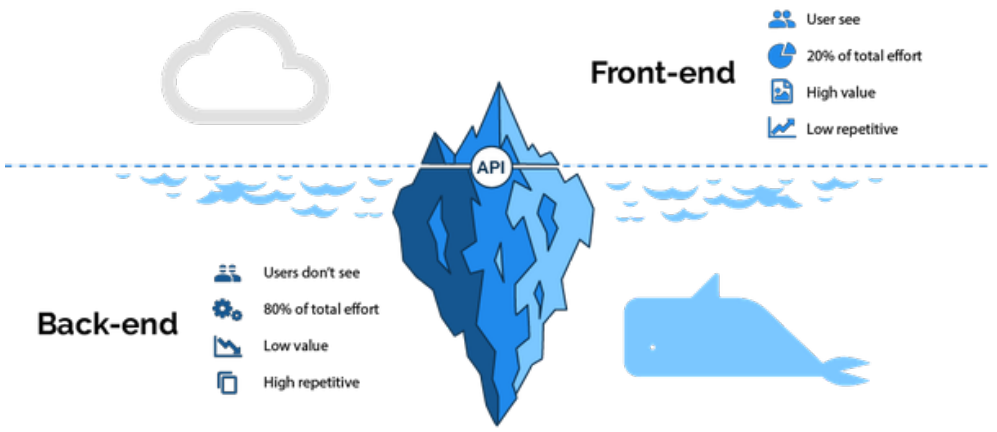

Figura 7.6: Representación del esfuerzo del *backend* frente al *front-end*.

7.5. Diagrama de despliegue de la aplicación

La aplicación web ha sido desplegada sobre un servidor nginx, aunque podría ser desplegado en un servidor con la tecnología Express de NodeJS perfectamente, o los servicios de hosting de Firebase. La tecnología de desarrollo de Node, permite realizar un "build", es decir una preparación para el despliegue generando los archivos necesarios.

Se ha tomado esta decisión de centralizar el servidor para facilitar la conexión con la red privada Ethereum del nodo de la máquina. De esta manera la máquina contendrá el nodo con todas las cuentas Ethereum del sistema, y gestionará los servicios de proveedor. Las cuentas tradicionales de usuario de Firebase, estarán enlazadas por tanto a la dirección Ethereum. La firma de cifrado de la cuenta, y de transacciones será compartida.

Además el servidor contendrá el sistema IPFS. De esta manera será privado y solo accesible por el sistema.

Básicamente tendríamos una arquitectura cliente servidor de tres niveles(Ver imagen 7.7, donde el servidor se comunica con los servicios distribuidos de Firebase, y con la *blockchain*, así como con IPFS.

Aunque inicialmente tomábamos una tecnología descentralizada, para lograr la mayor accesibilidad por parte del usuario sin requerir software de terceros, ha sido necesario una mayor centralización de los servicios para lograr una mayor seguridad.

Finalmente podemos ver el diagrama de despliegue de la aplicación en la figura 7.8

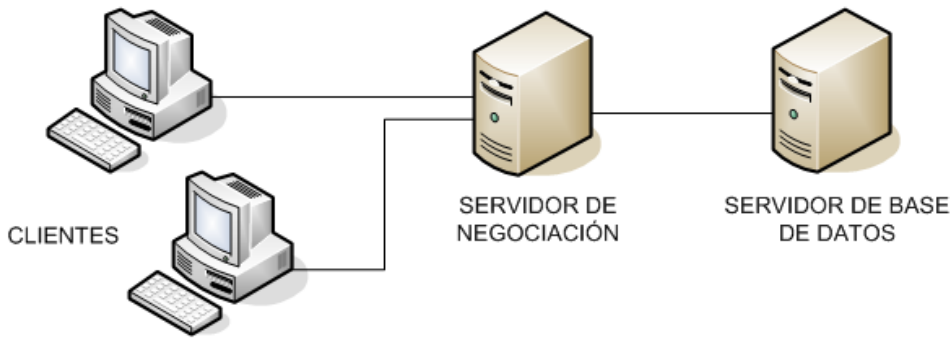

Figura 7.7: Cliente servidor tres niveles, tomada de [40]

7.6. Diseño de la base de datos

Como hemos dicho, se ha seleccionado los servicios de Firebase. En el caso del presente proyecto se ha seleccionado Firestore Real Time Database. Básicamente es una base de datos no relacional.

Partiendo del modelo conceptual realizado al comienzo del proyecto (Figura 5.1, se ha necesitado realizar una serie de cambios para adaptarse a una base de datos no relacional, así como se ha decidido realizar más cambios.

En la clase asociación del rol, pasa como atributo de grupo donde se almacena el nombre del usuario. Entrega pasa a ser una clase, donde la clase ficheros pasa a ser un array como atributo. Esta clase estará asociada únicamente a la tarea, y tendrá un atributo que almacene el grupo de la entrega.La asociación de responsable pasa a ser un atributo de tarea, que almacena la persona que la entregó.

Además debido a los cambios en las historias de usuario, se ha creado una nueva clase notificaciones. Los cambios se verán reflejados en la implementación final de la base de datos en la sección 8.3

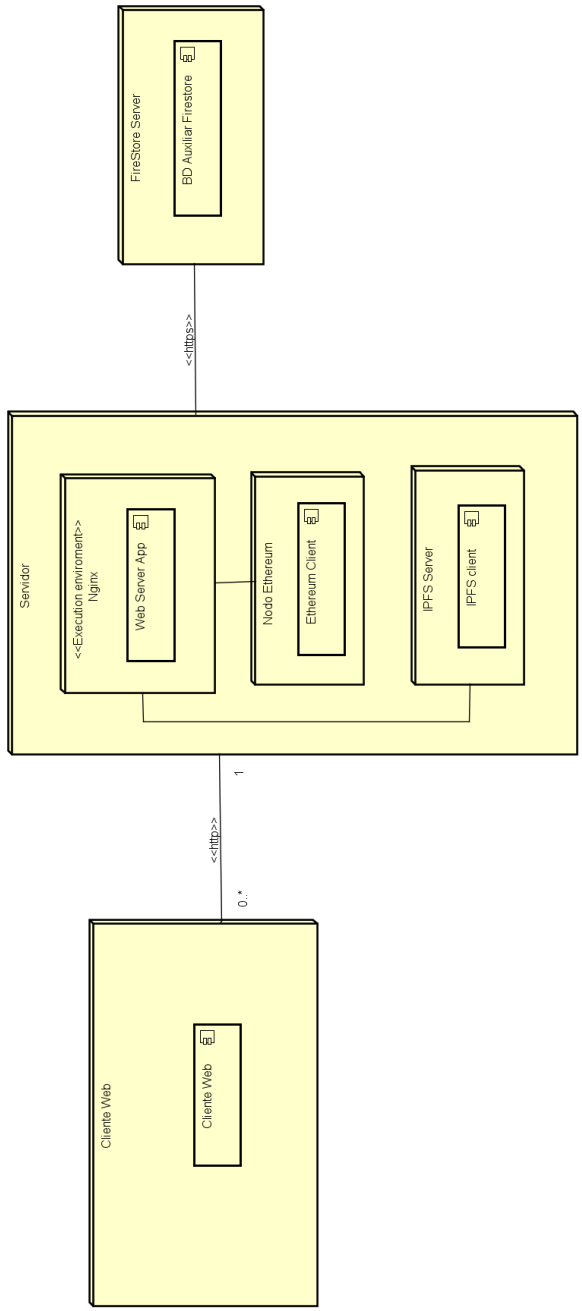

Figura 7.8: Diagrama de despliegue

Capítulo 8

Implementación, despliegue y pruebas

8.1. Implementación Ethereum

Para la implementación de GoEth es necesario realizar un script para minar solo cuando hay transacciones pendientes. El cliente por defecto mina bloques aunque no haya transacciones, de esta manera solucionamos el problema.

```
var mining_threads = 1

function checkWork() {
    if (eth.getBlock("pending").transactions.length > 0) {
        if (eth.mining) return;
        console.log("== Pending transactions! Mining...");
        miner.start(mining_threads);
    } else {
        miner.stop();
        console.log("== No transactions! Mining stopped.");
    }
}

eth.filter("latest", function(err, block) { checkWork(); });
eth.filter("pending", function(err, block) { checkWork(); });

checkWork();
```

Además es necesario implementar el primer bloque denominado "génesis". Este bloque

contiene la configuración de algunas de las reglas de la red Ethereum. Sirve para definir el gas por transacción, el id de la red o la dificultad de minado.

```
{
  "config": {
    "chainId": 2334156,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash": "0x0000000000000000000000000000000000...0",
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "clique": {
      "period": 16,
      "epoch": 30000
    }
  },
  "nonce": "0x0",
  "timestamp": "0x602bd66c",
  "extraData": "0x0020481527dba2d330b6c688d654c50086f917c9fa000...00",
  "gasLimit": "0x37d7840",
  "difficulty": "0x1",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x20481527dba2d330b6c688d654c50086f917c9fa",
  "alloc": {
    "20481527dba2d330b6c688d654c50086f917c9fa": {
      "balance": "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

8.1.1. Smart contract

Estructura del contrato inteligente utilizado para mantener el registro de las entregas y los archivos.

```
contract StorageEntrega {
  string public name = 'StorageEntrega';
```

```
  uint public fileCount = 0;

  //par key value
  mapping(uint => File) public files;


  struct File {
    uint fileId;
    string fileHash;
    string entregaHash;
    address payable uploader;
  }

  event FileUploaded(
    uint fileId,
    string fileHash,
    string entregaHash,
    address payable uploader
  );

  constructor() public {
  }

  function uploadFile(string memory _fileHash, string memory _entregaHash) public {

    require(bytes(_fileHash).length > 0);
    require(bytes(_fileName).length > 0);
    require(msg.sender!=address(0));

    fileCount ++;
    files[fileCount] = File(fileCount, _fileHash, _entregaHash, msg.sender);

    emit FileUploaded(fileCount, _fileHash, _entregaHash, msg.sender); // Emitir

  }
}
```

8.2. Estructura del código web

A continuación se muestra la estructura final del código del proyecto:

```
|- .gitignore
|- package.json
|- package-lock.json
|- truffle-config.js
|- README.md
|
```

```
|--- contracts
|    |- Migrations.sol
|    |- StorageEntrega.sol
|
|--- migrations
|    |- initial_migration.js
|    |- deploy_contracts.js
|
|--- public
|    |- index.html
|    |- manifest.json
|
|--- src
|    |- setupTests.js
|    |- reportWebVitals.js
|    |- App.js
|    |- App.test.js
|    |- App.css
|    |- index.js
|    |--- Assets
|              |-styles.css
|    |--- Componentes
|    |          |-PanelAdminGrupos
|    |          |                |- (omitido)
|    |          |-PanelAdminTareas
|    |          |                |- (omitido)
|    |          |-PerfilyGrupos
|    |          |                |- (omitido)
|    |          |-Genericos
|    |          |                |- (omitido)
|    |          |-Tareas
|    |          |                |- (omitido)
|    |          |- __tests__
|    |          |                |- (omitido)
|    |--- Contracts
|    |          |-Migrations.json
|    |          |-StorageEntrega.json
|    |
|    |--- Routers
|    |          |-PrivateRoute.js
|    |          |-PublicRoute.js
|    |          |-RouterJS.js
|    |          |-RouterMenuAdminGrupos.js
|    |          |-RouterPanelTareas.js
|    |          |-RouterMenuGrupos.js
|    |          |-RouterMenuTareas.js
|    |
|    |--- Servicios
|                  |---IPFS
|                  |      |- ipfs.js
|                  |      |- ipfsConnection.js
```

```
|                   |
|                   |---Web3JS
|                   |        |-Web3.js
|                   |
|                   |---FireBase
|                            |-fireconfig.js
|                            |-fireHooks.js
|                            |-fireRespaldo.js
|
|--- test
|    |-test.js
```

Los primeros ficheros que nos encontramos son ficheros de configuración de Node, Git, y de la herramienta Truffle. Después nos encontramos una serie de directorios:

- Contracts: Directorio que contiene los contratos que serán desplegados en la *blockchain*.
- Migrations: Scripts que permiten determinar la versión del contrato.
- public: Contiene el fichero index.html que es el punto de entrada de la aplicación cuando se realiza una petición hacia el servidor.
- src: Conocido como *source* contiene el código base de la aplicación
  - Componentes: Contiene los componentes de la aplicación descritos en 7.3.1. Además la carpeta __tests__ contiene los test de estos componentes.
  - Assets: Archivos auxiliares como imágenes o css.
  - Contracts: Contiene el despliegue de los contratos.
  - Routers: Componentes para la navegabilidad web
  - Servcios: Contiene las llamadas a la APIs, así como los hooks de estado que interaccionan con ellas.
- test: Contiene los test del *smart contract*.

8.3. Implementación del modelo

Al tratarse de una base NoSQL se ha necesitado realizar una serie de cambios a la hora de realizar la implementación, respecto a la idea inicial. Firestore trabaja con colecciones, que tienen a su vez documentos. Un documento sería por tanto cada uno de los elementos de la colección. El documento por tanto contiene los atributos establecidos, y pudiendo ser alguno incluso opcional.

Para mantener la coherencia y relación entre documentos, en Firebase existe un tipo de dato denominado "referencia". Una referencia apunta a un documento, de esta manera, pueden ser referenciados desde otras colecciones.

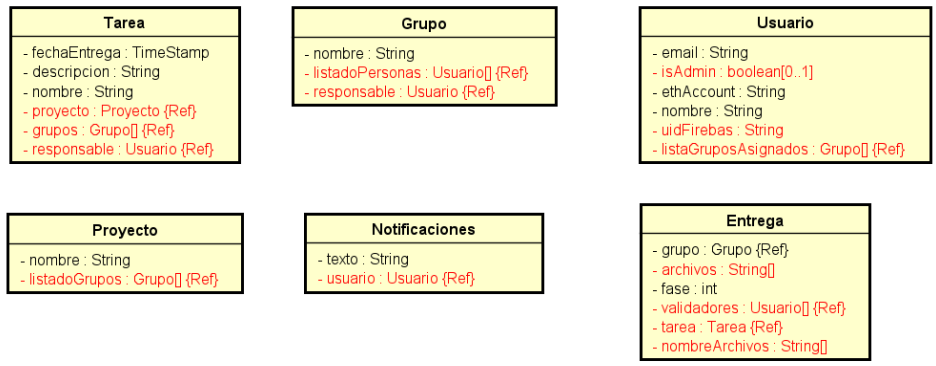

Figura 8.1: Modelo final en implementación

Los cambios finales se pueden ver en la figura 8.1.

Los cambios en rojos son aquellos derivados de la implementación. La restricción "Ref"significa por tanto que son referecias de firebase. El administrador ha sido implementado como un atributo, que puede existir o no. Si no existe básicamente significa que no es administrador.

8.4. Pruebas

En un comienzo se pensó en utilizar testing-library/jest-dom y testing-library como librerías para realizar test unitarios, que vienen por defecto React Create App para realizar los test.Sin embargo con la incorporación del Context de la aplicación de React para gestionar a los usuarios de Firebase [63] [58], y se implementó la librería 'Web3JS' y 'ipfs-http-client', se intentó mantener estas librerías para realizar los test unitarios, pero surgieron una serie de problemas. Estos problemas residen en que en estas librerías existe un conflicto, pues no gestiona correctamente los datos del useEffect proporcionados por el useContext que contiene el handler de Firebase (Se realiza un "Mock", es decir un objeto simulado, con el objetivo de suplantar los datos de la sesión y carga de datos) pues estos no son capaces de gestionar las dependencias de las variables a observar cuando se trata de una variable del contexto.

De esta manera para el testeo de We3bjs se ha utilizado Truffle. Así como los componentes de lista y formulario pertenecen a UI material, y por tanto están probados en cuanto a fiabilidad, y control de errores.

Para el resto se ha tratado de solventar el problema con las librerías de los test unitarios con test de caja negra por componente, así como se ha utilizado 'react developer tools' [48] para realizar un debug de la aplicación.

8.4.1. Tests de smart contract

Para el testeo de los smart contract nos apoyaremos en la herramienta de Truffle. Con el comando **truffle test ./path/to/test/file.js** podremos ejecutar aquel test que queramos. Es necesario el paquete mocha de npm a mayores. (npm install –save chai y npm install –save chai-as-promised)

El test desplegado en este caso para comprobar que funciona correctamente el smart contract sigue los siguientes pasos:

- Obtiene el código fuente del smart contract.
- Lo despliega en la red
- Comprueba que el *address* (Dirección pública) del contrato sea correcto.
- Se define un ejemplo de los parámetros que son necesarios para ejecutar el smart contract. A continuación este es ejecutado y se emite el evento asociado al contrato.
- Se comprueban por tanto que los valores de la ejecución del evento sean iguales a los que definimos en el ejemplo. Deben cumplirse estas afirmaciones.
- Añadimos los reject en caso de que los valores fuesen vacíos con el objetivo de obtener más información en caso de fallo.
- Se comprueba accediendo al bloque, y leyendo los datos, si estos son iguales.

```
const StorageEntrega = artifacts.require('./StorageEntrega.sol')

require('chai')
  .use(require('chai-as-promised'))
  .should()

contract('StorageEntrega', ([deployer, uploader]) => {
  let storageEntrega

  before(async () => {
    storageEntrega = await StorageEntrega.deployed()
  })

  describe('deployment', async () => {
    it('deploys successfully', async () => {
      const address = await storageEntrega.address
      assert.notEqual(address, 0x0)
      assert.notEqual(address, '')
      assert.notEqual(address, null)
      assert.notEqual(address, undefined)
    })
```

```
    it('has a name', async () => {
      const name = await storageEntrega.name()
      assert.equal(name, 'StorageEntrega')
    })
  })

  describe('file', async () => {
    let result, fileCount
    const fileHash = 'QmsSdudasadsu6n4Nrm45054ssdSDdsdrBCr739BN9Wb'
    const fileId = '1'
    const entregaHash = 'doc122355454'


    before(async () => {
      result = await storageEntrega.uploadFile(fileHash, entregaHash,
      { from: uploader })
      fileCount = await storageEntrega.fileCount()
    })

    //check event
    it('upload file', async () => {
      // El evento emitido debería ser igual al objeto inicial
      assert.equal(fileCount, 1)
      const event = result.logs[0].args
      assert.equal(event.fileId.toNumber(), fileCount.toNumber(), 'Id is correct')
      assert.equal(event.fileHash, fileHash, 'HashFile is correct')
      assert.equal(event.entregaHash, entregaHash, 'HashEntrega is correct')
      assert.equal(event.uploader, uploader, 'Address account is correct')

      // FAILURE: Sin file hash
      await storageEntrega.uploadFile('', entregaHash,
      { from: uploader }).should.be.rejected;

      // FAILURE: No hay id de la entrega.
      await storageEntrega.uploadFile(fileHash, '',
      { from: uploader }).should.be.rejected;


    })

    //Leer y comparar el objeto del bloque con el inicial
    it('file equal', async () => {
      const file = await storageEntrega.files(fileCount)
      assert.equal(file.fileId.toNumber(), fileCount.toNumber(), 'id is correct')
      assert.equal(file.fileHash, fileHash, 'Hash is correct')
      assert.equal(file.entregaHash, entregaHash, 'Hash is correct')
      assert.equal(file.uploader, uploader, 'Address account is correct')
    })
  })
})
```

8.4.2. Testeo final del front-end

Como hemos comentado, para sustituir el problema de los test unitarios se ha realizado un proceso de caja negra final donde se ha tratado de buscar los fallos no solventados dentro de los procesos de desarrollo de la aplicación.(En esta sección se mostrarán en forma de alto nivel) Las ventajas de los test de caja negra son :

- El programador y el probador tienen objetivos independientes.
- Desarrollo más rápido de las pruebas: No se requiere conocimiento y comprensión de la estructura interna, ya que solo se prueban las funciones externas del front-end.
- Sencillez: en el caso de aplicaciones grandes o complejas, la naturaleza inherente de las pruebas de caja negra ofrece una simplificación al verificar las salidas apropiadas recibidas en función de las entradas.

Desventajas

- Dado que no hay conocimientos sobre la implementación del código, el mismo código se puede probar varias veces, mientras que otras pueden no probarse nunca.
- Es imposible probar todas las entradas posibles en un período de tiempo razonable; por lo tanto, es posible que ciertos caminos nunca se prueben.

Test de navegabilidad

Nº	Descripción	Entrada url	Salida esperada	Resultado
P-01	Usuario no logueado intenta acceder a url protegidas	/grupos/*	Redirect login	Correcto
P-02	Usuario no logueado intenta acceder a url privada	/tareas/*	Redirect login	Correcto
P-03	Usuario sin rol de admin intenta acceder a url protegida	/panel-tareas/*	Redirect inicio	Correcto
P-04	Usuario sin rol de admin intenta acceder a url protegida	/panel-grupos/*	Redirect inicio	Correcto
P-05	Usuarios intentan acceder a tareas que no son de su grupo	/tareas/tarea-entrega/id	Redirect error	Correcto

Tabla 8.1: Batería de pruebas de alto nivel navegación

Test de creación de grupos, proyectos y tareas.

En estos test de alto nivel se busca testear la funcionalidad a la hora de la crear grupos, tareas o proyectos.

Nº prueba	Descripción de la prueba	Resultado
P-06	Prueba de introducción de campos vacíos en la creación de grupos, proyectos y tareas	Correcto
P-07	Grupos personalizados en los proyectos	Correcto
P-08	Grupos personalizados en las tareas	Correcto
P-09	Fecha correcta en la creación de las tareas	Correcto
P-10	Test de selección de responsable al crear la tarea	Correcto
P-11	Imposiblidad de crear una tarea antes de la fecha actual de hoy	Correcto

Tabla 8.2: Batería de pruebas de alto nivel creación de grupos, proyectos y tareas.

8.4.3. Test de modificación de grupos, proyectos y tareas

Se ha testeado la funcionalidad correspondiente a la modificación de las tareas, grupos y proyectos.

Nº prueba	Descripción de la prueba	Resultado
P-12	Modificación del responsable de grupo	Correcto
P-13	Modificación de los grupos de un proyecto	Correcto
P-14	Modificación de los miembros de un grupo	Correcto
P-15	Modificación de los datos de una tarea.	Correcto
P-16	Modificación de los datos de un grupo	Correcto
P-17	Modificación de los datos de un proyecto.	Correcto
P-18	Modificación del responsable de un grupo.	Correcto
P-19	Eliminación de grupo.	Error (Solucionado)
P-20	Eliminación de proyecto.	Correcto

Tabla 8.3: Batería de pruebas de alto nivel creación de grupos, proyectos y tareas.

Validación y visión de tareas

En esta sección se meustra una lista de las pruebas de alto nivel realizadas para la validación y visión de las tareas.

Nº prueba	Descripción de la prueba	Resultado
P-21	Solo responsable del grupo puede entregar	Correcto
P-22	Solo el responsable puede deshacer la entrega	Correcto
P-23	El usuario no puede volver a validad la tarea una vez hecho, y es informado de que la ha validado	Correcto
P-24	Cambio de fase correcto cuando todos validan la entrega	Correcto
P-24	El responsable no puede deshacer la entrega en fase 2	correcto
P-25	El administrador no puede validar hasta estar en fase 3.	Correcto
P-26	El administrador puede validar correctamente la tarea.	Correcto
P-27	Solo el responsable puede finalizar la tarea	Correcto
P-28	Se puede descargar el archivo en todas las fases tras la entrega por parte del	Correcto
P-29	Tarea visionada como finalizada si la fecha ya se ha superado.	Correcto
P-30	La tarea sigue activa para el último día de posibilidad de entrega.	Correcto

Tabla 8.4: Batería de pruebas validación de tareas

Batería de pruebas IPFS

Debido a que los *asserts* no funcionaban con la librería de IPFS, se ha comprobado manualmente la coincidencia del *hash* en IPFS y en el smart contract

Nº	Archivo	Hash	Coincide
P-32	prueba.pdf	QmdqHisEV24zkNDsBBg2BvvUt4td79FwH9t1hBFNzSt6ia	Sí
P-33	prueba.zip	QmVqNB8uazk9Fw8z6F3lrjNpDsTnrUT3V3R1jmDzh5a9oA	Sí
P-34	prueba.py	QmXhi1p5jfx8RhHpXbyPEZNkmaUoybG94byr6L9scqK5xv	Sí

Tabla 8.5: Batería de pruebas de alto nivel navegación

8.5. Despliegue de la aplicación.

En esta sección se desarrollará el proceso de despliegue de la aplicación.

8.5.1. Entorno de despliegue.

Se recomienda el despliegue de esta aplicación en redes privadas de internet, de hecho está pesando para ello, limitando su acceso al exterior para garantizar una mayor seguridad. En caso de realizar un despliegue de cara a la red pública, es necesario implementar el protocolo SSL sobre http, es decir, https.

En el presente documento se realizará de cara a la red pública, pues el objetivo es que sea utilizado por los alumnos de la universidad, sin tener que acceder vía VPN.

8.5.2. Herramientas requeridas

Las herramientas requeridas para el despliegue son:

- Node JS versión +14 [49]
- Npm versión 7+ [50]
- IPFS CLI [20]
- GoETH [33]

8.5.3. Proceso de preparación

- Tras la instalación de las herramientas anteriores es necesario obtener el código fuente del proyecto que está en la siguiente url `https://github.com/oxsaulxo/TFGBlockchain`.
- Una vez obtenido el código fuente es momento de instalar las dependencias del proyecto, para ello ejecutamos **npm install**
- Instalamos la herramienta Truffle con npm, para ello ejecutamos **npm install -g truffle –save**
- Una vez instaladas las dependencias crearemos un proyecto en firebase. `https://firebase.google.com/?hl=es`
- Inicializamos el daemon IPFS mediante en comando **ipfs init**. Se creará por defecto en el espacio de usuario una carpeta.

- A continuación necesitamos una serie de ficheros para el cliente geth. Los obtenemos desde la siguiente url.
  `https://drive.google.com/drive/folders/1AOchyg09b2LjvQksCNg4oqplBS1BgXsS?usp=sharing`
- Estos archivos serán guardados en un directorio fuera del directorio del código fuente del proyecto. En este directorio estará todo lo relacionado a Ethereum. Por ejemplo puede recibir este nombre ese directorio.

8.5.4. Proceso de configuración

Debido a que el plan es gratuito de firebase, no podemos exportar ni importar datos, por tanto habrá que realizar una serie de configuraciones a mano. El primer paso será coger los datos de la API que nos ha proporcionado al crear el proyecto dentro de firebase. Configuraremos los archivos fireconfig.js y fireRespaldo.js con estos datos que están situados en la ruta del proyecto `/src/Servicios/Firebase/`.

Como hemos dicho, no se puede importar manualmente, y por tanto no contamos con la configuración inicial de administrador. Por tanto, nos dirigimos al panel de firebase desde la web, y en authentificación creamos una cuenta en el Authentication ("Nueva cuenta", recordad este email y contraseña). Una vez creada, nos dirigimos a firestore, y creamos manualmente la colección Üsuarios". En esta colección, insertamos un nuevo documento con ID de documento automático y los siguientes campos de la tabla 8.5.4

Campo	**valor**	**tipo**
Email	El introducido anteriormente al crear la cuenta	String
Admin	true	Boolean
Nombre	Admin por ejemplo	String
Grupos	vacío	Array
Uid	Lo obtenemos en el panel de autentificación, del usuario creado	String
EthereumAccount	vacío	String

Tabla 8.6: Configuración inicial

A continuación nos dirigimos al directorio que creamos con nombre "Ethereum", y creamos un directorio denominado nodos dentro de él. Creamos entonces una cuenta para un minero, ejecutamos por tanto:
geth –datadir ./nodos account new
Y **introducimos la contraseña utilizada al crear la cuenta de Firebase anterior**, y nos quedamos con la dirección publica que nos indica. (Cuidado porque la puede indicar con todo en mayúsculas.. Podemos dirigirnos a dentro del directorio nodos, y en keystore obtenerla del nombre, o compararla para estar seguro)

A continuación nos vamos al fichero genesis.json, y editamos en el campo alloc la dirección que hay por la obtenido anteriormente (La actual es 0x20481527dba2d330b6c688d654c50086f917c9fa).

En este fichero además se puede configurar el id de la red. Nos dirigimos de nuevo a la colección Usuarios, y modificamos en campo EthereumAccount que estaba vacío introduciendo esta nueva dirección sin el 0x previo, por ejemplo '20481527dba2d330b6c688d654c50086f917c9fa'.

Inicializamos la *blockchain* con este primer bloque (Génesis.) Para ello ejecutamos el siguiente comando **geth –datadir ./nodos init ./genesis.json**

Ejecutamos la *blockchain* para ver los cambios (Introducir en network id el valor correspondiente si se ha variado la id de esta, cuidado con las comillas). parámetro console para lanzarlo en modo consola y poder ejecutar comandos:

```
geth --datadir ./nodos --networkid "2334156"
--http --http.api "eth,net,web3,personal,miner,admin" console
```

Consultamos que cuenta está situada como base (Es decir la que recibe el dinero al minar) Para ello ejecutamos **web3.eth.getCoinbase()** . En caso de no estar configurado, utilizamos **miner.setEtherbase(address)** donde address es la dirección pública que hemos obtenido del minero. Una vez configurado cerraremos el cliente geth.

Una vez configurado esto, iniciamos por primera vez IPFS, ejecutamos **ipfs daemon**, y paramos ipfs. Ejecutamos por tanto el siguiente comando para activar el cors: (cuidado con las comillas)

```
ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["*"]'
```

Configuración de ficheros

Primariamente hay que configurar el puerto donde se desplegará la red. En el archivo truffle-config.js de la raíz del proyecto configuramos los siguientes parámetros:

- Puerto: pondremos el puerto interno donde estará desplegado la red Ethereum
- Cambiar el id network si es necesario
- Mantenemos localhost, pues la red será privada.

Configuración de ficheros de conexión:

- `/src/Servicios/IPFS/ipfs.js` Configurar con la dirección pública ip, y el puerto público para la API. (Posteriormente habrá que mapear el puerto público al privado interno, se verá más adelante.)
  Si se quiere desplegar en local se utilizará `http://127.0.0.1:port` donde port será el puerto privado interno. (No será necesario el mapeo). En caso de la red privada `http://ipprivada:port`
- `/src/Servicios/IPFS/ipfsConnection.js` Configurar con la dirección pública ip, y el puerto público para el Gateway.(Posteriormente habrá que mapear el puerto público al privado interno, se verá más adelante.)
  Si se quiere desplegar en local se utilizará `http://127.0.0.1:port` donde port será el puerto privado interno. (No será necesario el mapeo) En caso de la red privada `http://ipprivada:port`

- `/src/Servicios/Web3/web3.js` Configurar con la dirección pública ip, y el puerto público para la api de web3js.(Posteriormente habrá que mapear el puerto público al privado interno, se verá más adelante.)
  Si se quiere desplegar en local se utilizará `http://127.0.0.1:port` donde port será el puerto privado interno. (No será necesario el mapeo) . En caso de la red privada `http://ipprivada:port`

Configuración de IPFS. Nos dirigimos a la carpeta donde está IPFS (Por defecto en el espacio de usuario, es una carpeta oculta) Allí habrá un archivo denominado config donde configuraremos el gatway y la API: (En el objeto adresses)

- gateway: `/ip4/0.0.0.0/tcp/puerto` donde puerto es el puerto interior del sistema.
- API: `/ip4/0.0.0.0/tcp/puerto` donde puerto es el puerto interior del sistema.

Es necesario por tanto mapear todo puerto publico con el privado

En el caso de red privada o local configurar la ip 0.0.0.0 con la ip correspondiente.

Compilación de contratos

Para compilar los contratos es necesario tener desplegada la red, por tanto la lanzamos con el comando que proporcionamos antes:

- Lanzamos la red Ethereum con la opción vista más atrás.
- Desbloqueamos la cuenta del minero con **web3.personal.unlockAccount('address',"password",0)**
- Mandamos al minero minar con miner.start(1)
- Compilamos con **truffle migration –reset**
- Paramos el minero con miner.stop()

8.5.5. Despliegue en servidor

En este caso se utiliza un servidor nginx. Se instala por tanto este tipo de servidor. (sudo apt-get install nginx)

En `/etc/nginx/sites-available` creamos un fichero de configuración: app.conf con el siguiente contenido

```
server {
        listen 80;
        server_name localhost;
        root /var/www/proyecto/build;
        index index.html;
```

```
        location / {

        add_header 'Access-Control-Allow-Origin' * always;
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Expose-Headers Content-Length;
        add_header Access-Control-Allow-Headers Range;
        try_files \$uri /index.html =404;
        }
}
```

- En listen ponemos el puerto privado interno donde está escuchando nginx. (Hay que mapear este puerto a una ip pública)
- En root indicamos la ruta donde está el build del proyecto.
- La configuración de location permite funcionar el CORS.

Eliminamos el enlace simbólico que hay en `/etc/nginx/sites-enables/default`, y creamos un enlace simbólico apuntando al fichero /etc/nginx/sites-available/app.conf desde `/etc/nginx/sites-enables`

Se genera el build del proyecto con npm run build.

Los resultados deben estar en /var/www/proyecto/build

8.5.6. Lanzamiento

Una vez todo preparado hay que lanzar el daemon de IPFS en segundo plano, ejecutamos: ipfs daemon

Tras esto lanzamos el cliente geth en segundo plano con el siguiente comando (Establecemos el puerto pertinente, -http addr 0.0.0.0 si se va a comunicar con el exterior.) El script detecta si existen transacciones y la mina en caso correcto.

```
geth --datadir ./nodos --networkid 2334156  --identity "testNet" --preload
"./mineWhenNeed.js"   --rpc.gascap '0' --rpc.txfeecap '0' --http --http.vhosts
"*" --http.port "4003" --http.corsdomain "*" --http.addr 0.0.0.0 --http.api
"eth,net,web3,personal,miner,admin" --allow-insecure-unlock console
```

A continuación desbloqueamos el minero desde la consola de Geth:

web3.personal.unlockAccount(.ªdress","password",0)

Y le mandamos minar con el siguiente comando.

miner.start(1)

Activar el servidor nginx.

8.5.7. Lanzamiento en local

Como hemos dicho, para lanzarlo en local se debe configurar con la ip de *localhost* o *loopback*, así como puertos internos.

Puede ser lanzado mediante npm como **npm run start**, con el daemon IPFS y geth lanzados.

8.6. Decisiones tomadas a lo largo del proyecto

Se han tomado la siguientes decisiones a lo largo del proyecto para cumplir con los objetivos.

- Para cumplir con el objetivo de establecer una mayor facilidad en la accesibilidad de la aplicación se ha decidido mantener todos las cuentas en un nodo. Si no cada usuario tendría que desplegar un nodo y configurarlo, utilizar algún proveedor como Metamask. Los usuarios serían responsables de la gestión de los nodos además, y de del mal uso de estas.
  Existía además otro problema relacionado a esto, y es que solo se permiten 25 conexiones remotas activas simultaneas si se hubiese utilizado ese modelo , lo que limitaría el acceso a la aplicación en caso de tener un número activos de usuarios.
- Se ha decidido utilizar Firebase para agilizar el proceso de desarrollo del *back-end*. Existía una tecnología de pago como era Spanner que proporcionaba una base de datos descentralizada. El hecho de centralizar ya el nodo de Ethereum en un servidor, ya rompía con la idea de la descentralización, por tanto, para facilitar el desarrollo del *back-end* y la gestión de usuarios se utilizó Firebase.
- La selección de Ethereum, frente a otras redes se basaba en la facilidad que otorgaba Ethereum para el desarrollo de aplicaciones
- IPFS era la única alternativa, y la más utilizada por la mayoría de las DApps. Swarm es una red P2P, y permitía el acceso de terceros a estos documentos.

8.7. Licencia

El software será propiedad de mis tutores, Joaquín Adiego, y Natalia Martín. Queda por tanto en sus manos la responsabilidad del desarrollo, mantenimiento, distribución y uso de la aplicación.

Capítulo 9

Seguimiento del proyecto

En el presente capítulo se abordarán los sprints realizados a lo largo del proyecto.

Cada historia de usuario del *Backlog* es transformada como un conjunto de subtareas que son realizadas en uno o en varios sprints con el objetivo de lograr dicha funcionalidad. A continuación en las diferentes secciones recogeremos los diversos sprints, donde en una tabla estarán las diversas tareas realizadas correspondientes a una o varias historias de usuarios. Se utilizará la siguiente nomenclatura:

- HU (Historia de usuario): Nº de la historia de usuario a la que está asociada una tarea. Si esa tarea no tiene que ver con una historia de usuario, es decir, no genera valor en el proyecto si no el equipo (Chore) se representará con ' - ' (Ya sea documentación, formación..). En muchos de los casos para no alargar las tablas se definirá únicamente la tarea, y no las subtareas que estaban a cargo, y si involucran a varias historias de usuario se indicará.
- TE (Tiempo empleado): Tiempo empleado medido en horas ininterrumpidas de trabajo
- Resultado: Indica el estado final de la tarea.

Cabe decir que inicialmente se estimaba realizar unas 33-34h por sprint. En ciertos puntos se han necesitado más horas para cumplir las tareas, incluso incumpliendo alguna y posponiéndola al siguiente sprint por una mala planificación

Recordemos que estamos empleando una estimación de *póker* para los *story points* donde respectivamente de 1 al 9, las horas que corresponden son: 1,2,3,5,8,13,21,34,55.

9.1. Sprint 0

Se ha realizado un sprint 0 con el objetivo de realizar un estudio previo de las tecnologías correspondientes a la *blockchain*. De esta manera se puede esclarecer la serie de dudas que había al respecto sobre esta tecnología, de manera que se puede seleccionar una tecnología en concreto. Por tanto obtenemos una arquitectura lógica inicial de la aplicación. (Ver tabla 9.1)

Este sprint además ha servido como redacción del capítulo 2 de la documentación.

Tarea	TE	Descripción	Resultado
T-01	2h	Definición inicial del *Backlog*	Completado
T-02	3h	Estructuración de la memoria	Completado
T-03	21h	Investigación de los conceptos de la *blockchain* y desarrollo de aplicaciones	Completado
T-04	4h	Investigación sobre las redes *blockchain*	Completado
T-05	5,5h	Investigación de aplicaciones basadas en Ethereum	Completado
T-06	2,5h	Modelo conceptual de dominio y arquitectura lógica	Completado
T-07	5h	Documentación capítulo introducción y estado del arte	Completado
T-08	8h	Planificación inicial del proyecto	Completado
Total	51h		

Tabla 9.1: Sprint 0

9.2. Sprint 1

En este sprint se ha realizado labores de documentación así como formación en React. En el caso de la formación en React se ha adquirido un curso de Udemy, así como se ha realizado otro que tenía el alumno que pudo obtener gratuitamente.

En cuanto a las estimaciones de *history points*:

- Se han estimado para la documentación y estructuración de la memoria 6 puntos que corresponden a 13h.
- Para la formación de React pues se conoce la duración se ha estimado 7 puntos pues es el rango en el cual entran dichos cursos. Corresponden 20h.
- Se ha estimado para el comienzo de la maquetación 4 puntos, pues inicialmente no se espera que este sprint aporte un incremento sustancial al valor del proyecto. Corresponde a 5h.

Sprint review: Al finalizar este sprint (Ver tabla 9.2) se ha corroborado con el *Scrum Master* la idea base del proyecto, con las diversas tecnologías que se han determinado. Por tanto inicialmente existía la idea de meter el documento en la *blockchain*, pero al ser computacionalmente imposible por el alto costo, reemplazándose por la inserción del *hash*.

Sprint planning: Se ha planificado para el siguiente sprint la maquetación para la gestión de grupos y proyectos, así como la funcionalidad correspondiente a estos apartados como para el *logueo* y registro de usuarios. Así como una primera versión de las tareas simple.

Además el alumno proseguirá con la formación en estas tecnologías.

HU	Tarea	TE	Descripción	Resultado
-	T-08	5h	Documentación tecnologías a utilizar	Completado
-	T-09	5h	Documentación planificación	Completado
-	T-10	2h	Estructuración del proyecto e instalación de software necesario	Completado
-	T-11	16,5h	Formación en React	Completado
HU-1	T-12	5h	Maquetación principal base	Completado
HU-1	T-13	2h	Maquetación estructura tareas y tarea	Completado
HU-1,2	T-14	3h	Maquetación listado de tareas	Completado
-	T-15	1h	Documentación sprint 0 y 1	Completado
	Total	44,5h		

Tabla 9.2: Sprint 1.

9.3. Sprint 2

En este sprint se han añadido dos nuevas historia de usuario, "listar grupos", y "listar proyectos", con el número 19, y 20 respectivamente (Ver tabla 3.2). Se ha añadido también la historia de usuario Çonsultar datos personales", con el número 21.

Se ha realizado una estimación para las siguientes tareas, superando un poco la idea inicial de las 33-34h:

- Registro y login de usuario 3 puntos que corresponden a 3h.
- Maquetación de perfil de usuario y información de grupo, 1 puntos (1h)
- Gestión de grupos maquetación y funcionalidad 4 puntos (5h)
- Maquetación y lógica de tareas 5 puntos (8h)
- Documentación plan de riesgos 3 puntos (3h)
- Formación en Web3JS y navegabilidad, 6 puntos (13h)

Sprint review: Tras finalizar este sprint (Ver tabla 9.3) he visto que la planificación ha sido bastante optimista. Pues realmente me he encontrado con problemas más de la propia librería de React, pues todavía al no tener gran destreza en el desarrollo ha limitado el progreso que inicialmente tenía previsto.

No se han podido iniciar los cursos que se tenían previsto, pues el desarrollo de la funcionalidad acordada ha sido lo primordial, y se decidido al finalizar la primera semana del sprint, posponer esto si fuese necesario. Finalmente tampoco ha sido posible finalizar la funcionalidad de *back-end* de las tareas.La funcionalidad lograda permite organizar a las personas en grupos, pudiendo modificar el responsable de este.

Sprint planning: Se ha planificado para el siguiente sprint por tanto la realización de estos dos cursos, así como se ha decidido estimar al alza los *story points*. En el siguiente sprint se centrará en finalizar la funcionalidad que ha quedado pendiente así como aplicar los conocimientos adquiridos en los cursos que se realicen, haciendo énfasis en la seguridad de la navegación para limitar las rutas a los que puede acceder un usuario. Se pretende además poder gestionar y crear proyectos.

HU	Tarea	TE	Descripción	Resultado
HU-7,16	T-16	9h	Maquetación login y register, lógica front-end errores, *back-end* usuarios y *hooks* firebase. Modificar rol de usuario a administrador.	Completado
HU-2,21	T-17	1h	Maquetación mi perfil, mi grupo y submenu superior, y lógica de *back-end*	Completado
HU-10,11,19	T-18	9h	Maquetación añadir grupo, modificación de grupo,listado de grupos y submenu grupos. Lógica de *back-end*	Completado
HU-8,9	T-20	3h	Maquetación listado de tareas admin, añadir tarea,modificar tarea, submenu admin	Completado
HU-07,16	T-21	1,5h	Pruebas funcionalidad usuarios	Completado
HU-07,16	T-22	1,5h	Pruebas funcionalidad *back-end* grupos	Completado
-	T-23	4h	Documentación plan de riesgos	Completado
-	T-24	-	Formación librería Web3JS	Sin iniciar
-	T-25	-	Formación routers y rutas privadas.	Sin iniciar
HU-12,18,20	T-26	-	Lógica *back-end* añadir tareas	Sin iniciar
	Total	29h		

Tabla 9.3: Sprint 2

9.4. Sprint 3

Este sprint por tanto como se comentó al finalizar el anterior ha servidor para retomar las tareas que no pudieron ser iniciadas, y ampliar la funcionalidad existente.

Asignación de los *story points*:

- Formación en Web3JS y navegabilidad, 6 puntos (13h)
- Documentación capítulo 5, 3 puntos (3h)
- Creación de cuentas Ethereum 5 puntos (8h)
- Navegabilidad web 4 puntos (5h)
- Añadir tareas 2 puntos (2h)
- Añadir crear y gestionar proyectos 4 puntos (5h)

Sprint Review: (Ver tabla 9.4) Han surgido bastantes problemas a la hora de implementar los sub-routers. Actualmente existen nuevas versiones referentes al router, que son diferentes a las utilizadas en los vídeos de formación . Por tanto, esto ha generado que tuviese que consultar la documentación para realizar varias modificaciones para adaptar los routers y los subrouters.

En cuanto a la implementación de la librería Web3JS, de creación cuentas también surgió un problema con el CORS que finalmente pudo ser solucionado. Era necesario ejecutar el cliente con un parámetro en concreto. Por tanto aunque el tiempo estimado ya era alto, los problemas ocasionados hasta dar solución a ello ha extendido bastante el tiempo empleado en esa tarea, y en el testeo de esta implementación.

Todos estos problemas han ocasionado que la funcionalidad relacionada con los proyectos no haya podido ser implementada, y solo lo relacionado con la maquetación. Tampoco se ha podido implementar la lógica de añadir tareas.

Como consejo del *Scrum Master*, debería haber completado la funcionalidad de añadir tareas, pues estaba sin iniciar en el anterior sprint.

Sprint planning: En el próximo sprint se ha planteado finalizar la funcionalidad que queda pendiente, incluyendo la funcionalidad de listar tareas para los usuarios.

9.5. Sprint 4

El presente sprint tiene como objetivo realizar las tareas atrasadas, así como la funcionalidad relacionada con las tareas

- Añadir, y gestionar proyectos: 5 puntos (8h)
- Añadir tareas: 5 puntos (8h)
- Listar tareas, y navegabilidad y restricciones a ellas: 6 puntos (13h)
- Mejoras en la gestión de grupos y proyectos (Prediseñados): 4 puntos (5h)

HU	Tarea	TE	Descripción	Resultado
-	T-27	4h	Formación routers, subrouters y rutas privadas	Completado
-	T-28	7,5h	Formación librería Web3JS	Completado
-	T-29	2,5	Documentación capítulo 5	Completado
HU-17,7	T-30	9h	Cerrar sesión, proveedor Web3JS, cuentas asignadas a la cuenta firebase, desbloqueo de cuentas Ethereum.	Completado
HU-16	T-31	8h	Implementación de routers, y subrouters	Completado
HU-16,7	T-32	6h	Pruebas navegabilidad routers y Web3JS	Completado
HU-12,18,20	T-33	2h	Maquetación crear proyecto, modificación de proyecto y listado	Completado
HU-12,18,20	T-34	-	Lógica back-end proyectos	Sin iniciar
HU-12,18,20	T-35	-	Lógica back-end añadir tareas	Sin iniciar
	Total	40h		

Tabla 9.4: Sprint 3

Sprint Review: (Ver tabla 9.5) Se ha realizado la funcionalidad atrasada. Sin embargo, se han encontrado una serie de escollos relacionados a la hora de mostrar las tareas que tiene asignadas el usuario en función del grupo, pues existía un pequeño *bug*, que correspondía a que si una persona estaba en dos grupos de un mismo proyecto solo aparece una de las opciones.

El problema ha sido localizado y es relacionado con las promesas. Existe una lista de referencias de Firebase de cada grupo, y hay que realizar este conjunto de peticiones, y por tanto el problema es que el estado de React se actualiza antes de que finalicen las promesas.

Sprint planning: Se ha propuesto una solución mandando esperar todas esas promesas para la tarea incompleta. La idea es finalizar en el próximo sprint este error, así como la funcionalidad relacionada con la eliminación de grupos y tareas, y parte del administrador que permite modificar tareas. Por cuestiones laborales, este sprint será más corto.

HU	Tarea	TE	Descripción	Resultado
HU-12,18,20	T-36	7h	Añadir proyectos, eliminar proyectos, listas y edición. (Funcionalidad back-end también)	Completado
HU-08	T-37	2h	Back end añadir tareas a proyectos.	Completado
HU-19	T-38	16h	Lógica front-back end para listar tareas y enlaces dinámicos a tareas concretas.	Sin finalizar
HU-19	T-39	4h	Pruebas de errores en listar tareas en función del grupo , en y las tareas asignadas al grupo.	Completado
HU-11,18	T-40	3h	Grupos personalizados en proyectos y tareas	Completado
	Total	32h		

Tabla 9.5: Sprint 4

9.6. Sprint 5

Como hemos dicho en el anterior sprint planning, este sprint será más corto por temas laborales. Por tanto los puntos de historia correspondiente a las tareas:

- Solución a los errores y finalización de la historia H-20, 5 puntos (8h)
- Eliminación de grupos, 2 puntos (2h)
- Eliminación de tareas, 1 punto (1h)
- Modificación de tareas, 2 puntos (2h)
- Documentación de los sprints realizados, 3 puntos (3h)
- Listado de entregas, 2 puntos (2h)
- Realizar entrega 2 puntos (2h)

Sprint Review: (Ver tabla 9.6) Se ha implementado un *hook context* de estado con el objetivo de mantener en el estado de la aplicación información relacionada con la sesión del usuario, como los grupos en los que está asignado. De esta manera reduciremos ciertas consultas. Se ha solucionado además los problemas que existían con la lista, mediante un await all de todas las promesas.

Para eliminar los grupos requería de eliminar cierta información en otras colecciones. He tardado más de lo planeado, pero finalmente se ha completado la tarea. Los problemas residían en el acceso mediante los tipos de referencia de Firestore, que a la hora de realizar ciertas condiciones como consulta no conseguía realizarlo correctamente.

Sprint planning: En el próximo sprint el objetivo es comenzar la funcionalidad de las entregas relacionado con la *blockchain* y con IPFS

HU	**Tarea**	**TE**	**Descripción**	**Resultado**
HU-19	T-41	6h	Solución errores en el context (Hook) de la aplicación en el listado de tareas y petición dinámica de promises.	Completado
HU-19	T-42	2h	Test de errores en el listado de tareas	Completado
HU-13	T-43	5h	Funcionalidad eliminar grupos	Completado
HU-13	T-44	5h	Pruebas eliminación de grupos	Completado
HU-14,9	T-45	1h	*Back-end* eliminar y modificar tareas	Completado
HU-4	T-46	1h	Realizar entrega *back-end* Firestore	Completado
HU-4,15	T-47	1h	Listado de entregas admin, modificación funcionalidad	Completado
-	T-48	3h	Documentación sprints 2,3,4 y 5	Completado
	Total	24h		

Tabla 9.6: Sprint 5

9.7. Sprint 6

Ha existido un problema con mi ordenador personal, y he tenido que posponer este sprint prácticamente 1 mes. Para cuadrar las fechas, se ha decidido comenzar en la fecha del que sería el 8º. Por

tanto la fecha límite del proyecto sería de cara a la convocatoria extraordinaria.

El objetivo de este sprint es implementar el sistema de entregas. Para ello implementaremos el sistema IPFS, así como los contractos inteligentes. También se desplegará las herramientas para desarrollar los contratos inteligentes, así como la lógica de back end de las tareas. Se realizará también un script para controlar el minado.

- Formación IPFS, 4 puntos (5h)
- Añadir archivos a IPFS, 5 puntos (8h)
- Descargar archivos de IPFS, 5 puntos (8h)
- Desarrrollo de smart contracts, 4 puntos (5h)
- Despliegue de smart contracts y herramientas, 1 punto (1h)
- Script de minado, 2 puntos (2h)
- Ejecución de smart contracts y obtener datos de los bloques, 3 puntos (3h)

Sprint Review: (Ver tabla 9.7) Se ha realizado una formación en IPFS. Posteriormente se ha realizado la funcionalidad para almacenar ficheros en este sistema, así como para descargarlos. En el caso de esto segundo, ha sido un poco complejo porque es almacenado con el nombre del hash que genera. Por tanto ha habido que almacenar el nombre del fichero, e utilizar una librería auxiliar para permitir la descarga correcta del documento.

Además se ha realizado el contrato y desplegado el contrato para la inserción de las tareas en la *bockchain*.

Por tanto también se ha realizado la lógica de back-end para las entregas.

Algunas tareas han quedado incompleta.

Sprint planning: En la próximo sprint se finalizarán las tareas que han quedado incompletas relacionadas con almacenar la entrega en la *blockchain*, así como se realizará el sistema de validación.

HU	**Tarea**	**TE**	**Descripción**	**Resultado**
-	T-49	5h	Formación IPFS API	Completado
-	T-50	1h	Configuración IPFS y herramientas de smart contracts	Completado
HU-4	T-51	6h	Desarrollo de smart contracts, y migrations	Completado
HU-4	T-52	2h	Test truffle smart contract	Completado
HU-4,15	T-53	1,5h	Funcionalidad de almacenar el archivo a IFPS	Completado
HU-4,15	T-54	19h	Funcionalidad descarga del fichero subido a IPFS.	Completado
HU-4	T-55	7h	Pruebas y solucción a errores IPFS	Completado
HU-4	T-56	-	Script minado cuando	No iniciado
HU-4	T-57	-	Subir el archivo a la *blockchain*	No iniciado
	Total	41,5h		

Tabla 9.7: Sprint 6

9.8. Sprint 7

En este sprint por tanto se realizarán las tareas relacionadas con la implementación del sistema de fases descrito en la sección 5.3. Básicamente primero implementaremos la funcionalidad para insertar el *hash* del documento de IPFS en la *blockchain* junto con los demás datos de la entrega, y también implementaremos los métodos para leer este *hash* de la *blockchain* y para obtener el documento. Además se realizará un script para controlar el minado para cuando solo existan transacciones.

Utilizaremos posteriormente esta funcionalidad en el sistema de validación que hemos citado anteriormente por fases. Se implementará por tanto la funcionalidad requerida para que los miembros validen la tarea.

Además se ha decidido también posteriormente realizar una serie de cambios visuales.

- Ejecutar transacciones y leer bloques de la *blockchain*, 3 puntos (3h)
- Script de minado, 3 puntos (3h)
- Sistema de validación, 7 puntos (20h)
- Eliminar entrega, puntos 2 (2h)
- Cambios visuales, puntos 4 (5h)

Sprint Review: (Ver tabla 9.8) Se han completado todas las tareas dentro del tiempo establecido al comienzo del proyecto, que debería emplearse para realizar un sprint. Por tanto se ha implementado toda la funcionalidad requerida para validar las tareas, y almacenarlas en la *blockchain*.

Sprint planning: El Scrum Master ha propuesto realizar una nueva serie de nuevas funcionalidades: Notificaciones para el proceso de validación y la entrega múltiple. Por tanto

9.9. Sprint 8

Se han añadido dos nuevas historias de usuario: Entrega múltiple (22) y notificaciones de usuario (23). Se realizarán las tareas acordes a esta funcionalidad, así como tareas acordes a la gestión del balance de las cuentas de Ethereum. Por último se realizarán tareas relacionadas con la documentación

- Entrega múltiple 6 puntos (13h)
- Maquetación y funcionalidad de notificaciones 6 puntos (13h)
- Gestión de balances de cuentas 2 puntos (2h)
- Documentación diseño, implementación y revisiones. 5 puntos (8h)

Sprint Review: (Ver tabla 9.9) Se han encontrado impedimientos en la funcionalidad correspondiente a la entrega múltiple debido a las restricciones que impone la API Web3JS,la *blockchain* y el sistema IPFS. Por lo demás la funcionalidad correspondiente a las notificaciones fue realizada correctamente y sin problemas y en un tiempo bastante corto y menor de lo previsto. Se realizaron tareas de documentación

Sprint planning: Se ha descartado la funcionalidad de la entrega múltipe para el siguiente sprint.Por tanto se realizarán tareas correspondientes a la documentación, y solución a posibles errores, y mejoras visuales.

HU	Tarea	TE	Descripción	Resultado
HU-4	T-58	1h	Script minado	Completado
HU-4,	T-59	5h	Almacenar el *hash* del documento y la entrega en la *blockchain*. Lectura del bloque de la *blockchain*, para obtener el hash del documento de la entrega	Completado
HU-4	T-59	1h	Solución a problemas de gas de la red. Reconfiguración de la red.	Completado
HU-4,15,5	T-60	18h	Implementación completo del sistema de validación por fases.	Completado
HU-4,5	T-61	3h	Pruebas fases de validación	Completado
HU-4	T-62	1h	Eliminar entrega	Completado
HU-19,20,2,6	T-63	1h	Modificación del componente lista, por componente de lista UI material.	Completado
HU-2	T-64	1h	Maquetación y lógica de lista de tareas finalizadas	Completado
HU-7,4	T-65	2h	Cambios en el *front-end*, componente barra de carga asíncrona	Completado
-	T-66	1h	Cambios visuales en los colores de la aplicación	Completado
	Total	34h		

Tabla 9.8: Sprint 7

9.10. Sprint 9

Este es el último sprint, donde se desarrollarán tareas de documentación, y pequeños cambios en la aplicación

- Despliegue de la aplicación: 5 puntos (8h)
- Cambios visuales de la aplicación: 2 puntos (2h)
- Test y correción de bugs: 1 punto (1h)
- Documentación despliegue: 3 puntos (3h)
- Documentación sprints: 3 puntos (3h)
- Documentación análisis final de costes: 1 punto (1h)
- Modificación y documentación conclusiones, resumen y abstract: 1 punto (1h)
- Documentación, manual de usuario: 3 puntos (3h)
- Documentación pruebas: 2 puntos (2h)
- Revisión final de la memoria: 5 puntos (8h)

Sprint Review: (Ver tabla 9.10) Se da por concluido el proyecto en este sprint con el despliegue de la aplicación y una serie de cambios visuales. Por lo demás se ha producido una redacción de la memoria de aquellos puntos que quedaban por realizar. Se han solucionado algunos bugs a la hora

HU	Tarea	TE	Descripción	Resultado
HU-22	T-67	20h	Entrega múltiple	Descartada HU
HU-23	T-68	3h	Maquetación y funcionalidad de notificaciones	Completada
HU-7	T-69	1h	Balance de cuentas ilimitado	Completada
HU-23	T-70	1h	Pruebas notificaciones	Completada
-	T-71	4h	Documentación diseño	Completada
-	T-72	1,5h	Documentación implementación	Completada
-	T-73	1,5h	Cambios documentación tecnologías utilizadas	Completada
-	T-74	1,5h	Cambios documentación scrum	Completada
	Total	33,5h		

Tabla 9.9: Sprint 8

de eliminar un grupo que había problemas con la recursividad de la eliminación, así como había un problema con el número de los validadores de un grupo por tomar mal el nombre de la variable.

En cuanto al despliegue, nunca había realizado uno real, por tanto tuve que investigar para realizar todo correctamente. Se ha utilizado un servidor nginx sobre las máquinas virtuales de la uva.

HU	Tarea	TE	Descripción	Resultado
-	T-75	4h	Despliegue en la máquina virtual	Completado
HU-4,5	T-76	1	Cambios visuales en la aplicación	Completado
HU-4	T-77	1h	Solución bug validadores	Completado
HU-13	T-78	1h	Solución bug eliminar grupo	Completado
-	T-79	2,5h	Documentación despliegue	Completado
-	T-80	0,5h	Documentación análisis final de costes	Completado
-	T-81	0,5h	Documentación decisiones en el proyecto	Completado
-	T-82	0,5h	Documentación resumen y abstract	Completado
-	T-83	0,5h	Documentación conclusiones	Completado
-	T-84	3h	Pruebas finales	Completado
-	T-85	2h	Documentación pruebas	Completado
-	T-86	2h	Documentación manual de usuario	Completado
-	T-87	2h	Documentación sprint 6,7,8 y 9	Completado
-	T-88	3h	Revisión final de la memoria y pequeños cambios	Completado
-	T-87	0,25h	Reajuste sesión de usuario	Completado
	Total	24,75		

Tabla 9.10: Sprint 9

Capítulo 10

Conclusiones y trabajo futuro

10.1. Conclusiones finales sobre el proyecto

Cuando se comenzó este proyecto existían una serie de objetivos técnicos y personales. El resultado ha sido satisfactorio en cuanto a ellos.

En cuanto a los objetivos técnicos de la aplicación de la sección 1.2.1:

- Se ha realizado un estudio para determinar las tecnologías, siendo este satisfactorio, y con buen resultado para el desarrollo del proyecto.
- Se ha podido aplicar una arquitectura basada en *blockchain* por tanto a la hora de realizar el proyecto.
- Se ha realizado un sistema de validación basado en fases con el objetivo de dar soporte a las necesidades.
- Se ha desarrollado la aplicación sin requerir de la instalación de un software de terceros.

En cuanto a los objetivos personales referentes a la sección 1.2.2:

- Se han adquirido conocimientos en desarrollo de front-end en un nuevo *framework* (Aunque React se define como una librería). Tenía conocimiento en tecnologías más viejas, y de esta manera se han actualizado mis conocimientos de cara al mercado laboral
- Se han adquirido conocimientos acerca de una tecnología que en el futuro podría ganar mucha fuerza como es la *blockchain* de cara al registro de transacciones.
- He ampliado mis conocimientos acerca de la planificación de proyectos gracias a estas dos vías. Tanto en las prácticas como en este proyecto, se han aplicado la metodología Scrum.
- Finalmente se han cumplido también los objetivos de ganar autonomía en el trabajo, frente a posibles problemas donde no abundasen soluciones, así como se ha conseguido realizar un despliegue real como adquirir conocimientos en un lenguaje nuevo como Solidity.

10.2. Funcionalidad futura de la aplicación

En un futuro la aplicación pudiera requerir de mejoras:

- Migración a una base descentralizada e implementación de los servicios de backend.
- Mejoras visuales en los formularios de la página web.
- Permitir el envío de notificaciones de los administradores hacia los usuarios.
- Generación de notificaciones vía email también.

Apéndice A

Manual de usuario

En este apéndice se describirá el proceso de creación de grupos, proyectos, y tareas desde el punto de vista del administrador, hasta llegar a la validación de tareas correspondiente a la funcionalidad del usuario común. A.5 De esta manera servirá como manual de usuario para ambas partes este proceso de descripción de la funcionalidad web.

Antes de nada vamos a explicar la funcionalidad de la barra superior del apartado del administrador. Esta barra la podemos ver por ejemplo en la figura A.3. Básicamente un usuario común tiene los apartados de tareas (Donde está la lista de tareas y la entrega y la validación) y mi grupo (Donde hay información relacionada con el usuario.), así como un apartado del manual de usuario de la página, el icono para ir a sus notificaciones y logout. A mayores por tanto un administrador tiene:

- Panel de tareas: Donde tiene la funcionalidad para crear y modificar tareas, así como para crear y modificar proyectos.
- Panel de grupo: Donde tiene la funcionalidad para crear grupos y registrar usuarios.

Figura A.1: Barra administrador

Figura A.2: Barra usuario

A.1. Creación de usuarios

Las personas que tiene rol de administrador son los únicos que pueden registar usuarios en la aplicación.

Hay que tener en cuenta que este proceso de creación tardará un poco pues hay que crear su cuenta ethereum. Por tanto, esta contraseña que se proporcione servirá de cifrado de la cuenta y no es modificable. Podemos ver este proceso con la retroalimentación en las figuras A.3 y A.4

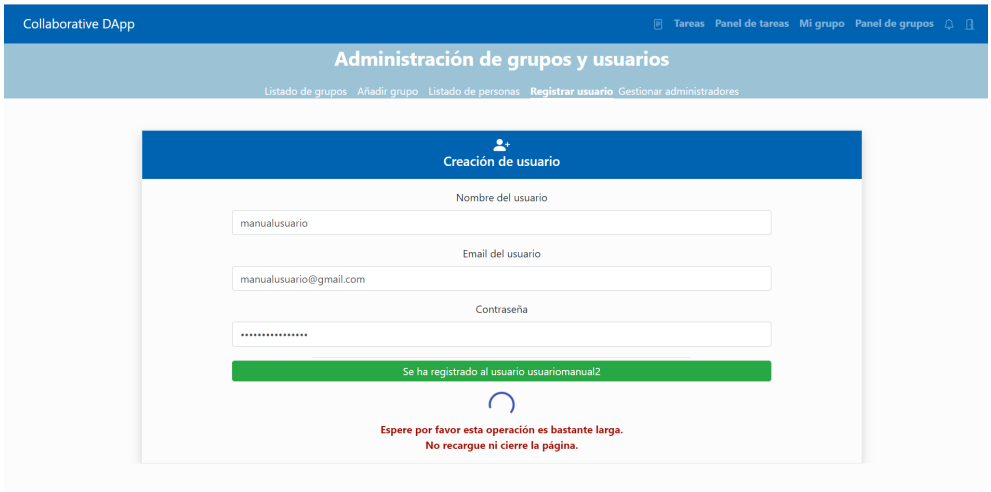

Figura A.3: Creación de cuentas

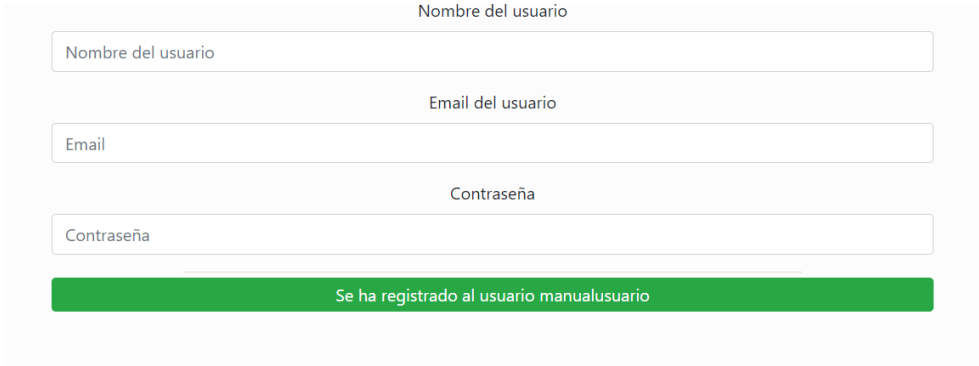

Figura A.4: Creación de cuentas

A.2. Creación de grupos

La creación de grupos es parte de la funcionalidad también del administrador. Básicamente el administrador selecciona en el buscador/selector a lo usuarios que quiera añadir, y posteriormente selecciona un responsable para el grupo, que será la persona que realizará las entregas, y las finalizará.

Podemos ver el proceso en las figuras A.5 y A.6

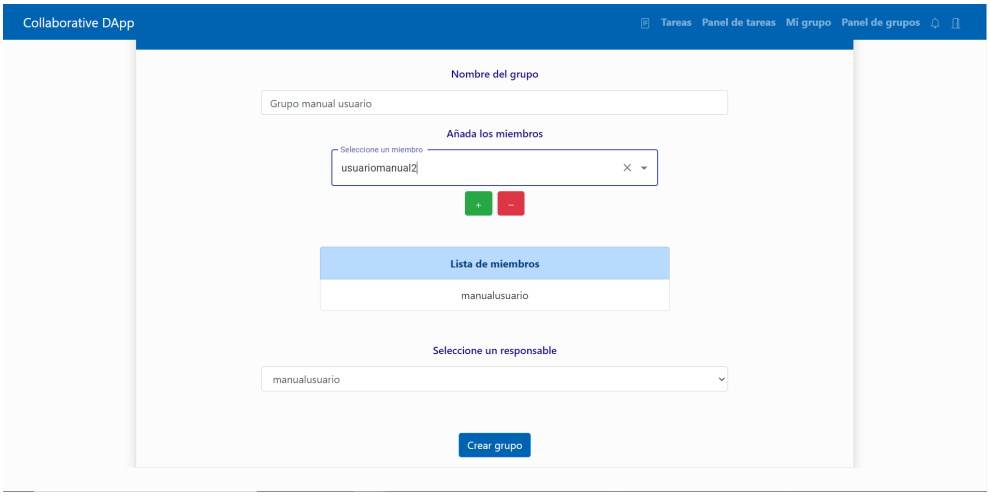

Figura A.5: Creación de grupos

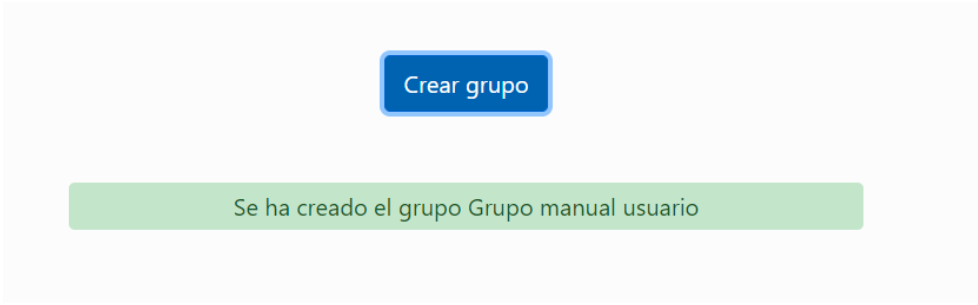

Figura A.6: Creación de grupos

A.3. Creación de proyectos

Otra funcionalidad del administrador es la creación de proyectos. Hay que recordar que a estos proyectos se le asignan tareas. En la creación de proyectos podemos añadir todos los grupos posibles, o bien que sea personalizado y seleccionar aquellos que se desee. Podemos ver esto en la figura A.7

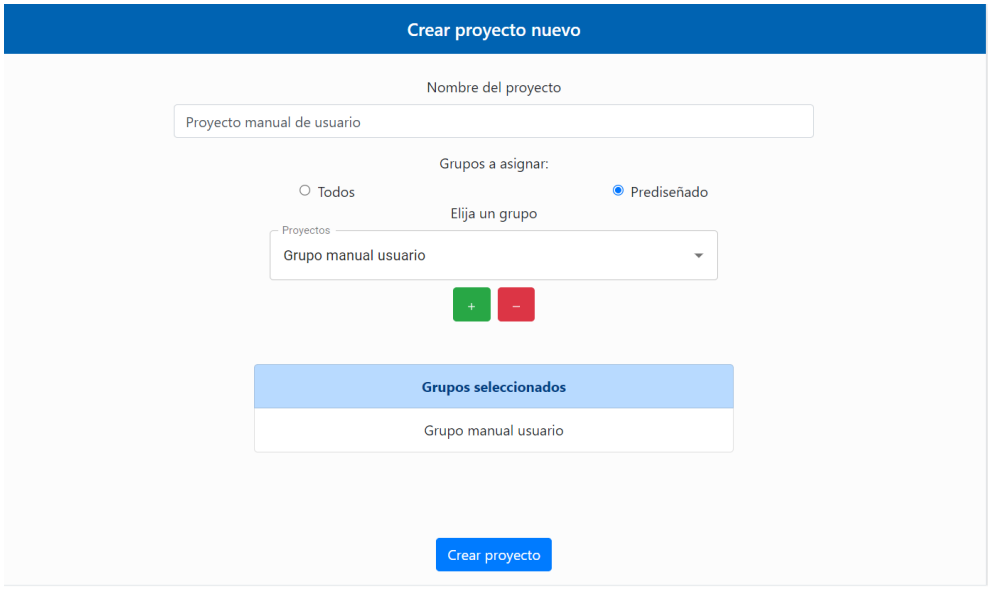

Figura A.7: Creación de proyectos

A.4. Añadir tareas

Como hemos dicho anteriormente, el administrador puede añadir tareas. Se asigna esta tarea a un proyecto, y dentro de este proyecto se puede asignar esta tarea a todos los grupos que forman ese proyecto, o de manera personalizada aquellos grupos o grupo que queramos. A.8

Además contiene los siguientes campos:

- Fecha fin de la tarea, en la cual se marca como finalizada para todos los grupos.
- Nombre de la tarea
- Descripción de la tarea

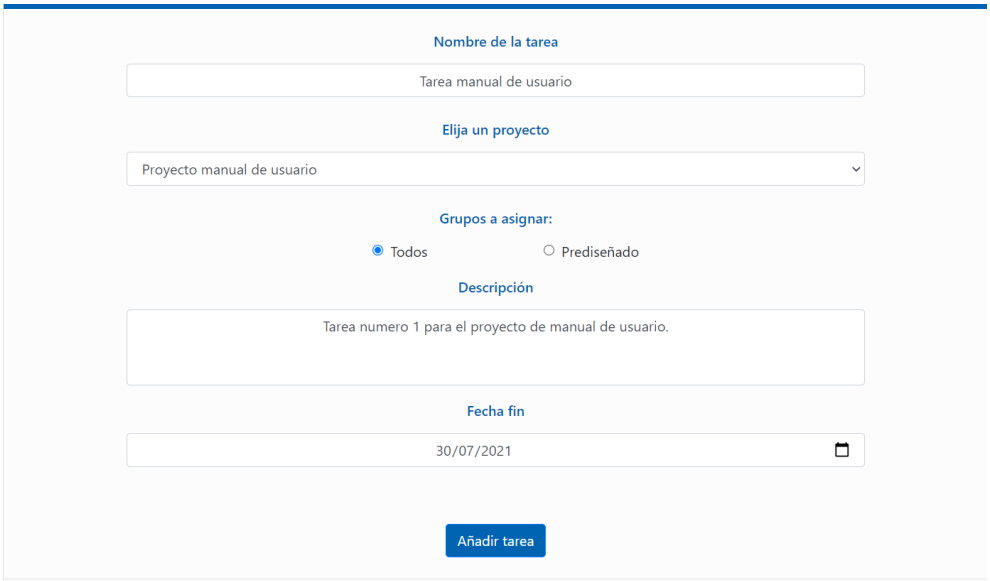

Figura A.8: Añadir tareas

A.5. Validación de tareas

A.5.1. Fase de entrega

El responsable del grupo debe subir el documento. Podemos ver este proceso de subida en la figura A.10

En el caso de no ser el responsable se indicará en un recuadro que el usuario espere a que el responsable realice la entrega. Podemos verlo en la figura A.9

Una vez que sube el documento, el responsable puede deshacer siempre y cuando todos los miembros del grupo no hayan validado la entrega. A.11

Figura A.9: Fase de entrega

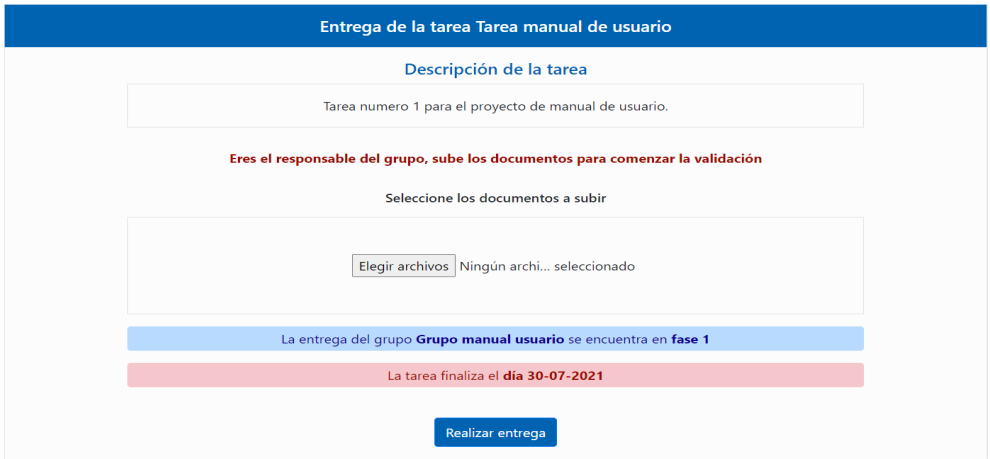

Figura A.10: Fase de entrega

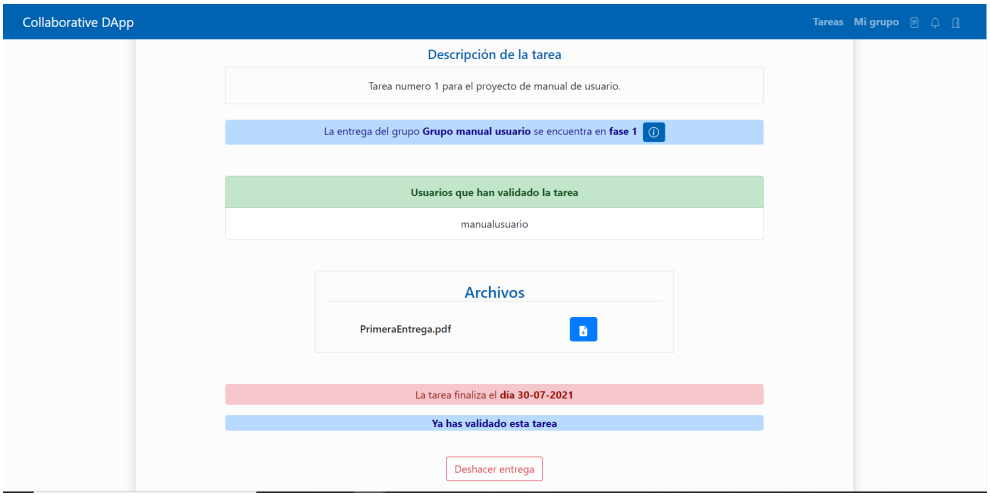

Figura A.11: Fase de entrega

A.5.2. Validación de grupo

Comienza por tanto el proceso de validación del grupo. Cada miembro deberá por tanto validar esta tarea.

- Cada uno de los miembros del grupo podrá descargar el fichero/s correspondientes a la entrega realizada por el responsable.
- Si está todo correcto validará la entrega.
- Si algo no está correcto, puede hablar con el responsable de la entrega para que deshaga la entrega y por tanto realice de nuevo otra entrega y recomience otra vez este proceso de validación.
- Una vez que todos han validado la entrega, se pasa de fase y no se puede deshacer la entrega.

Existe un botón al lado del mensaje de la fase, que sirve como ayuda de recuerdo de qué partes involucra cada fase.

Podemos ver este proceso en las figuras A.12 y A.13

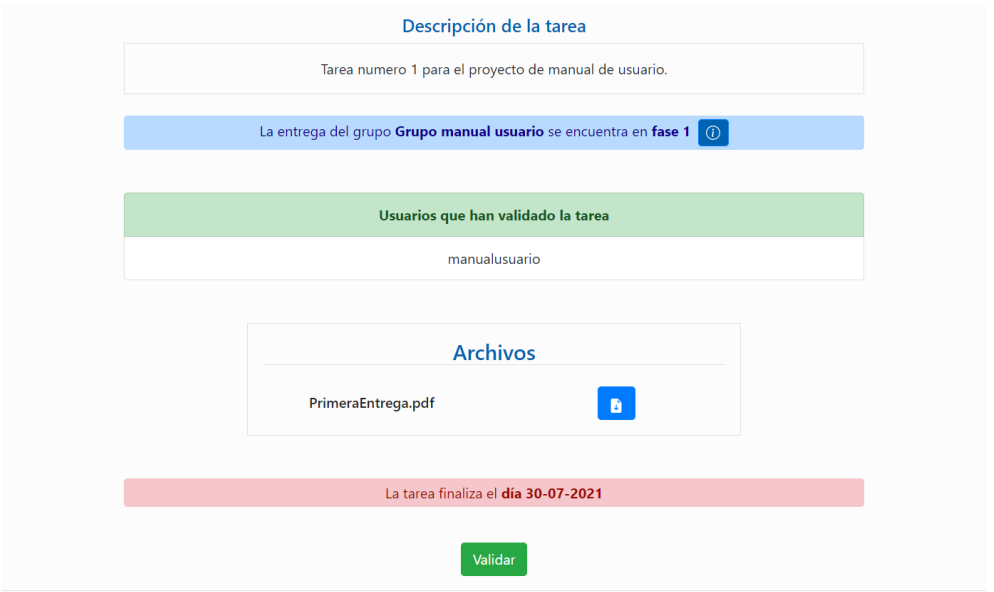

Figura A.12: Validación de grupo

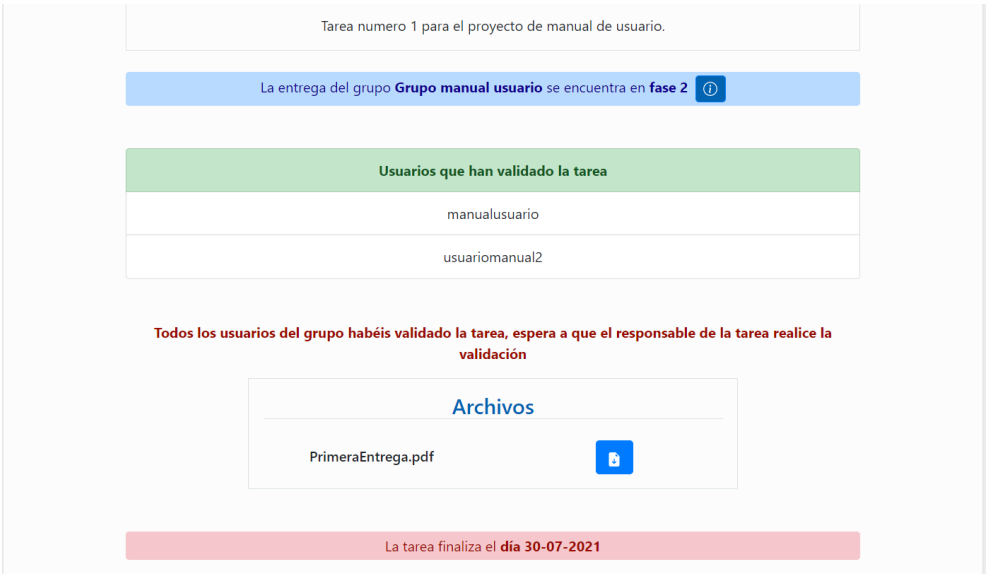

Figura A.13: Validación de grupo

A.5.3. Validación creador de la tarea

Una vez que el grupo valida la tarea, el administrador tiene que validar esta entrega. Por tanto, desde el panel de tareas, en el apartado de gestionar tareas (Tiene dos opciones para cada tarea, editarla o ver las entregas) irá a la tarrea correspondiente para ver las entregas. A.14

En la lista de entregas A.15 se puede ver las diversas entregas que ya hay, y la fase en la que se encuentra cada una.

Finalmente podemos ver en la A.16 y A.17 este proceso de validación por parte del administrador, donde también puede descargar los archivos asociados.

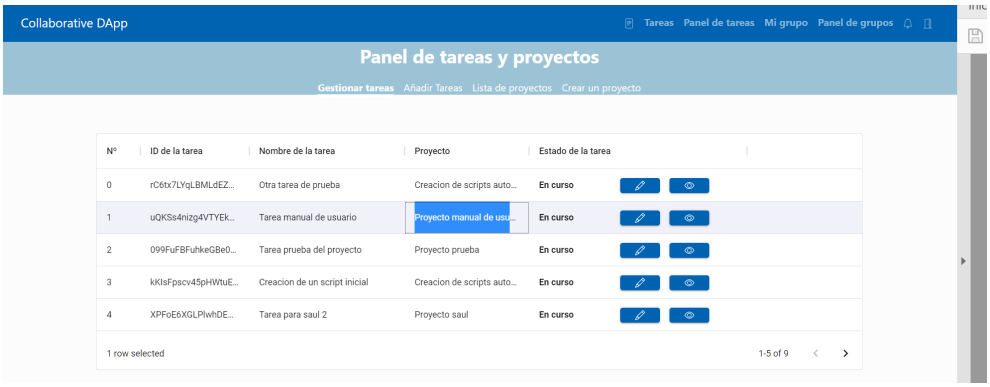

Figura A.14: Validación creador de la tarea

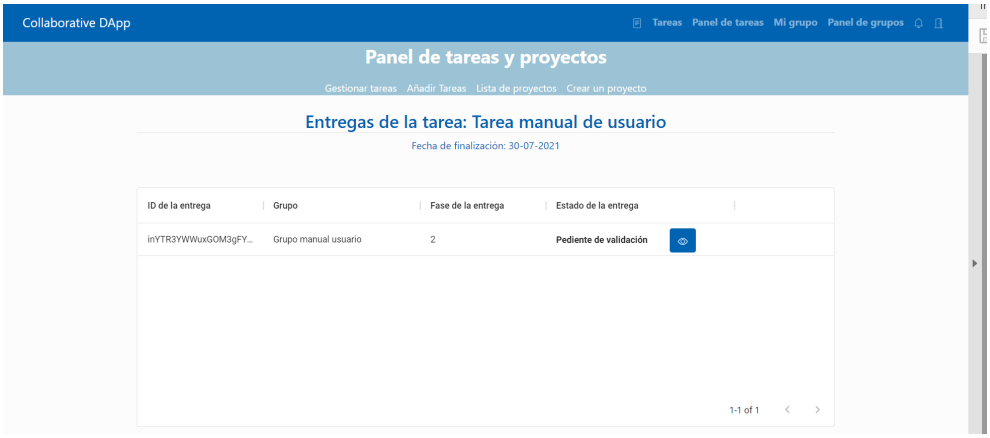

Figura A.15: Validación creador de la tarea

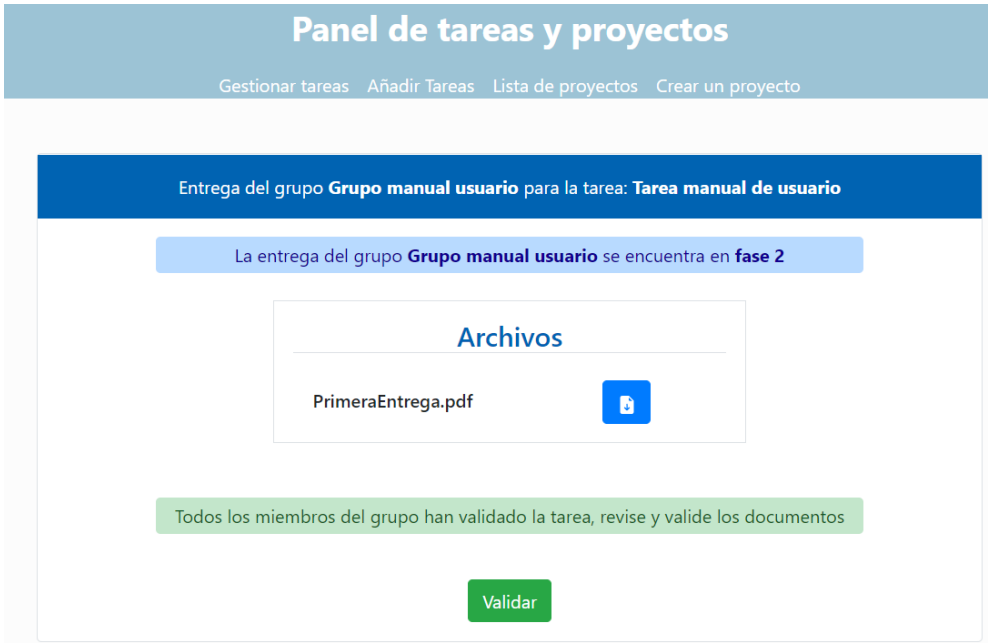

Figura A.16: Validación creador de la tarea

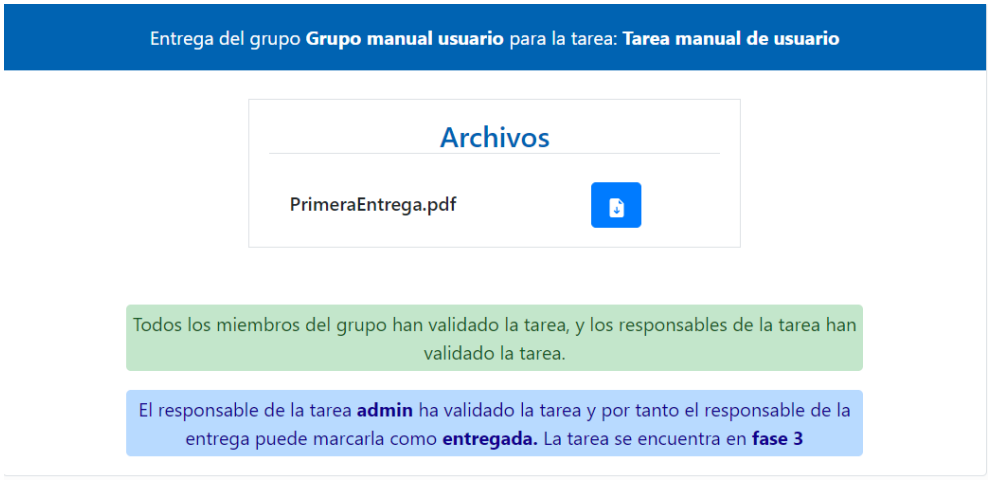

Figura A.17: Validación creador de la tarea

A.5.4. Finalizar tarea

Una vez producido la validación por parte del administrador, se produce la finalización de la entrega. Este proceso tarda bastante, pues se envía una transacción a la blockchain para que sea minada.

Podemos ver este proceso en las figuras A.18 y A.19

Figura A.18: Finalizar tarea

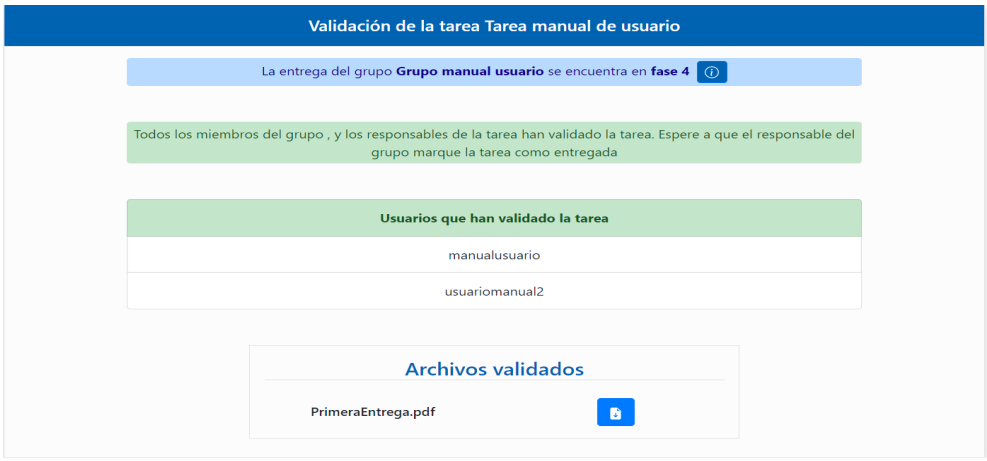

Figura A.19: Finalizar tarea

A.6. Edición grupos, tareas y proyectos

El administrador puede modificar los grupos, proyectos y tareas existente. En las diversas subsecciones veremos como.

A.6.1. Edición de proyectos

En la figura A.20 podemos ver la lista de proyectos existente, así como el número de grupos que tiene asociados. Podemos editar este proyecto, y como vemos en la figura A.20 se pueden añadir y quitar grupos, así como actualizar el nombre del proyecto o eliminar este proyecto.

Figura A.20: Edición de proyectos

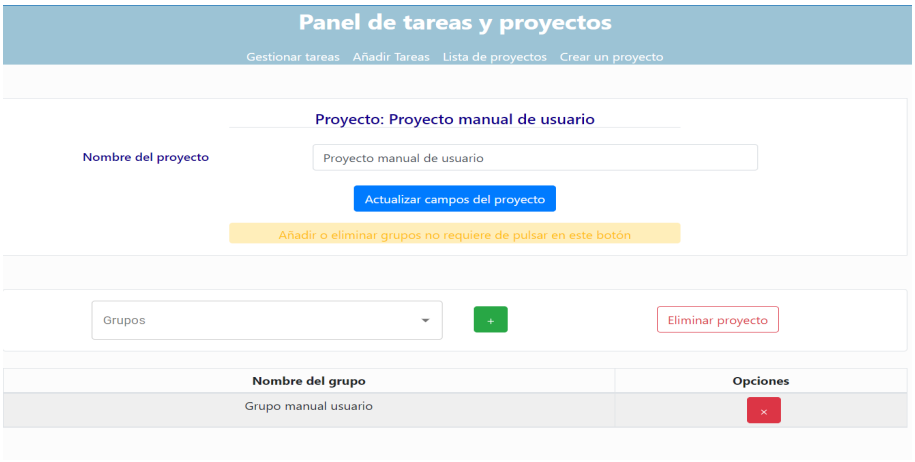

Figura A.21: Edición de proyectos

A.6.2. Edición de grupos

El administrador por tanto también puede modificar los grupos listados en la figura A.22. Como podemos ver en la figura A.23 se pueden quitar y añadir personas al grupo. Además se puede cambiar el responsable del grupo. Como solo hay un responsable de grupo, la otra persona perderá dicho rol, es decir, se produce una transferencia de los poderes.

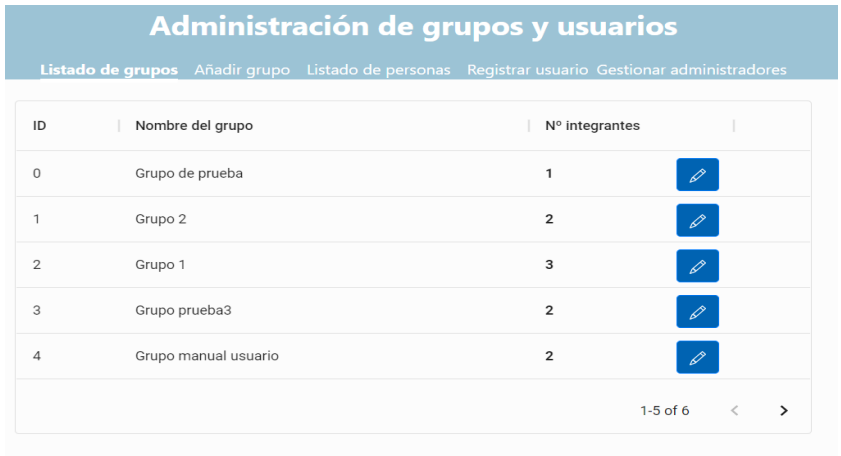

Figura A.22: Edición de grupos

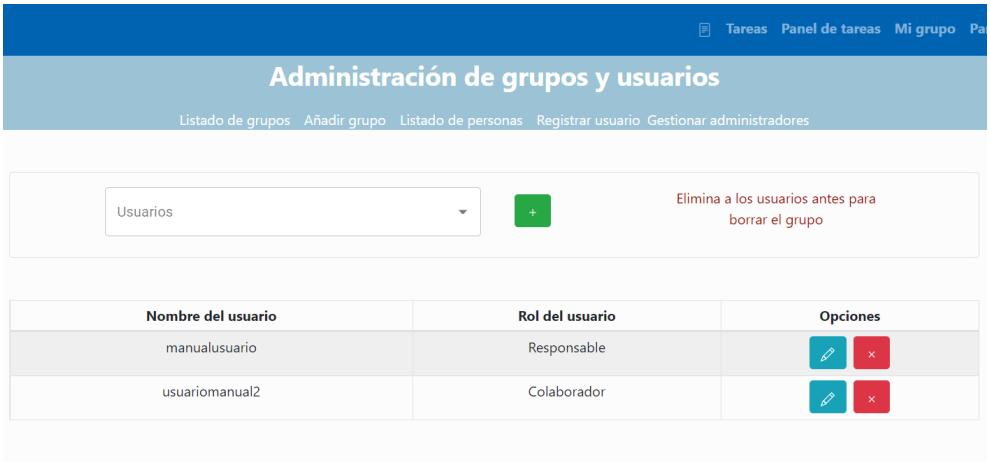

Figura A.23: Edición de grupos

A.6.3. Edición de tareas

El administrador puede editar también las tareas existentes. Puede cambiar los datos de la tarea, o añadir nuevos grupos que pertenezcan al proyecto al que está asignado en esa tarea. Podemos ver esto en más detalle en la figura A.24

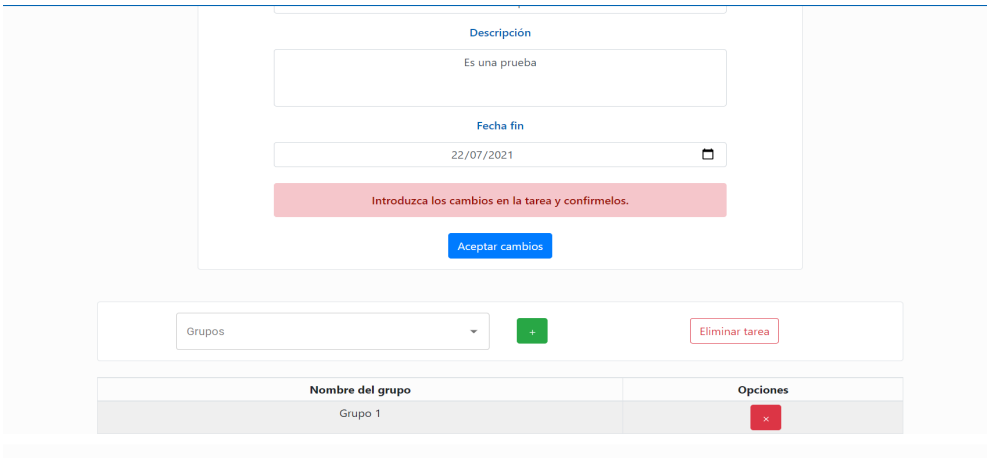

Figura A.24: Edición de tareas

A.7. Mis notificaciones

Todos los usuarios cuentan con notificaciones que son generadas cuando se producen eventos como la entrega de la tarea por parte del responsable del grupo o de la tarea.

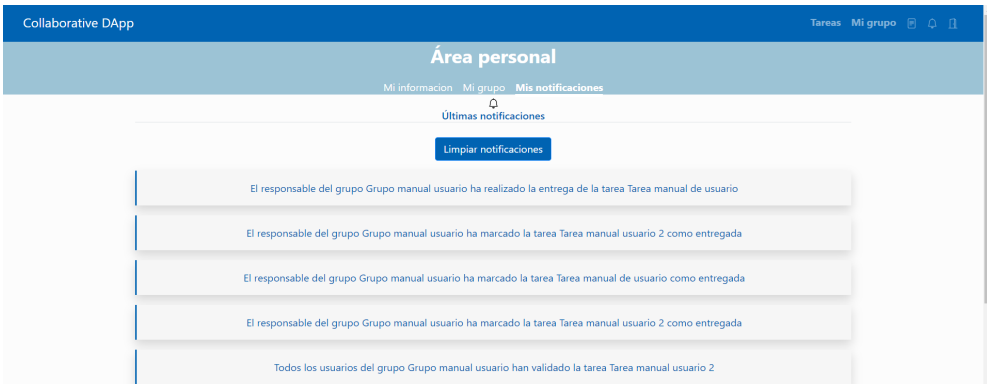

Figura A.25: Mis notificaciones

Apéndice B

Resumen de enlaces adicionales

Enlaces adicionales de este Trabajo de Fin de Grado:

- Repositorio con el código fuente `https://github.com/oxsaulxo/TFGBlockchain`
- Recursos adicionales de configuración y scripts: `https://drive.google.com/drive/folders/1AOchyg09b2LjvQksCNg4oqplBS1BgXsS`

Bibliografía

[1] *¿Cómo se crea un bloque?* es. Feb. de 2020. URL: https://academy.bit2me.com/mineria-bitcoin-como-se-crea-un-bloque/ (visitado 07-02-2021).

[2] *¿Qué es la dificultad de minería en Bitcoin?* es. Dic. de 2019. URL: https://academy.bit2me.com/que-es-dificultad-mineria-bitcoin/ (visitado 07-02-2021).

[3] *¿Qué es npm?* es. Ene. de 2021. URL: https://www.freecodecamp.org/espanol/news/node-js-npm-tutorial/ (visitado 10-07-2021).

[4] *¿Qué es Scrum?* en. URL: https://www.scrum.org/resources/blog/que-es-scrum (visitado 05-02-2021).

[5] *@material-ui/core*. en. URL: https://www.npmjs.com/package/@material-ui/core (visitado 10-07-2021).

[6] *5 Git Workflows & Branching Strategy to deliver better code*. en. Mayo de 2020. URL: https://zepel.io/blog/5-git-workflows-to-improve-development/ (visitado 23-02-2021).

[7] Bit2Me Academy. *¿Qué es la recompensa de bloque?* es. Jun. de 2019. URL: https://academy.bit2me.com/que-es-recompensa-de-bloque/ (visitado 10-02-2021).

[8] *Aplicaciones descentralizadas (dapps)*. es. URL: https://ethereum.org (visitado 11-02-2021).

[9] *Ataque del 51 %*. en. Jul. de 2018. URL: https://komodoplatform.com/en/blog/51-attack-how-komodo-can-help-prevent-one/ (visitado 11-02-2021).

[10] Atlassian. *Jira — Software de seguimiento de proyectos e incidencias*. es. URL: https://www.atlassian.com/es/software/jira (visitado 11-07-2021).

[11] *Atomic Design*. en. Jun. de 2013. URL: https://bradfrost.com/blog/post/atomic-web-design/ (visitado 11-07-2021).

[12] *Backend como servicio: ¿Qué es un BaaS?* es-ES. Ene. de 2021. URL: https://blog.back4app.com/es/que-es-un-baas-backend-como-servicio/ (visitado 25-05-2021).

[13] Vlad Balin. *React MVX*. Nov. de 2020. URL: https://github.com/gaperton/React-MVx (visitado 25-05-2021).

[14] *Blockchain Networks for DApps*. Dic. de 2020. URL: https://blaize.tech/services/how-to-choose-the-right-blockchain-network-for-your-dapp-development/ (visitado 13-02-2021).

[15] *Blockchain: Desarrollando Smart Contracts para Ethereum con Truffle*. es. Mayo de 2019. URL: https://www.returngis.net/2019/05/desarrollando-smart-contracts-para-ethereum-con-truffle/ (visitado 10-07-2021).

[16] *BOE.es - BOE-A-2021-2292 Resolución de 4 de febrero de 2021, de la Dirección General de Trabajo*. URL: https://www.boe.es/eli/es/res/2021/02/04/(3) (visitado 07-07-2021).

[17] *Centralizado, Descentralizado y Distribuido: Diferencias*. es. URL: https://latinotoken.com/conceptos-basicos-101-centralizado-descentralizado-distribuido-p2p-cuales-las-diferencias/ (visitado 11-02-2021).

[18] *Centralized vs Decentralized: Core Differences?* en-US. Ago. de 2018. URL: https://101blockchains.com/centralized-vs-decentralized-internet-networks/ (visitado 11-02-2021).

[19] Alistair Cockburn. *Agile Software Development*. Highsmith Series, 2002. URL: https://archive.org/details/agilesoftwaredev0000cock (visitado 05-02-2021).

[20] *Command-line*. en-US. URL: https://docs.ipfs.io/install/command-line/ (visitado 12-07-2021).

[21] Luke Conway. *La blockchain*. en. URL: https://www.investopedia.com/terms/b/blockchain.asp (visitado 14-02-2021).

[22] *Crear una nueva aplicación React – React*. es. URL: https://es.reactjs.org/docs/create-a-new-react-app.html.

[23] davidbritch. *Modelo Model-View-ViewModel: Microsoft*. es-es. URL: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (visitado 25-05-2021).

[24] *Design Patterns for Blockchain*. URL: https://www.researchgate.net/publication/342963667_Emerging_Design_Patterns_for_Blockchain_Applications (visitado 25-05-2021).

[25] *El concepto de consenso en Blockchain, su relevancia y consecuencias*. es-ES. Jul. de 2019. URL: https://santanderglobaltech.com/concepto-consenso-blockchain-relevancia-consecuencias/ (visitado 11-02-2021).

[26] *Estimación del costo del proyecto*. es. Nov. de 2014. URL: https://www.recursosenprojectmanagem
com/estimacion-del-coste-del-proyecto/ (visitado 11-07-2021).

[27] *Ethereum Apps You Can Use Right Now*. URL: https://consensys.net/blog/news/90-ethereum-apps-you-can-use-right-now/ (visitado 11-07-2021).

[28] *Flowchart Maker & Online Diagram Software*. URL: https://app.diagrams.net/ (visitado 11-07-2021).

[29] *GitHub: ¿Qué Es GitHub Y Cómo Utilizarlo?* es. Abr. de 2019. URL: https://www.hostinger.es/tutoriales/que-es-github (visitado 23-02-2021).

[30] *GitHub: Where the world builds software*. en. URL: https://github.com/ (visitado 23-02-2021).

[31] *Gnosis*. URL: https://gnosis.io/ (visitado 13-02-2021).

[32] *Go Ethereum*. URL: https://geth.ethereum.org/ (visitado 10-07-2021).

[33] *Go Ethereum*. URL: https://geth.ethereum.org/ (visitado 12-07-2021).

[34] *Guía de scrum*. en. URL: http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf (visitado 05-02-2021).

[35] *Guía para estimar usando story points*. es. URL: https://es.linkedin.com/pulse/gu%5C%C3%5C%ADa-para-estimar-usando-story-points-roberto-a-r-moran (visitado 05-02-2021).

[36] *Hashing*. en-US. URL: https://docs.ipfs.io/concepts/hashing/ (visitado 11-07-2021).

[37] *How IPFS works*. en-US. URL: https://docs.ipfs.io/concepts/how-ipfs-works/ (visitado 11-07-2021).

[38] Trey Huffine. *componentDidMakeSense — React Component Lifecycle Explanation*. en. Dic. de 2019. URL: https://levelup.gitconnected.com/componentdidmakesense-react-lifecycle-explanation-393dcb19e459 (visitado 18-02-2021).

[39] Bob Hughes y Mike Cotterell. *Software Project Management*. 5a edición. Europa, Oriente Medio y Africa: McGraw-Hill Education, 2009. ISBN: 9780077122799.

[40] *Imagen servidor tres niveles*. es. URL: https://es.wikipedia.org/wiki/Programaci%5C%C3%5C%B3n_por_capas (visitado 25-05-2021).

[41] *IPFS Documentation*. en-US. URL: https://docs.ipfs.io/ (visitado 10-07-2021).

[42] *ipfs-http-client*. en. URL: https://www.npmjs.com/package/ipfs-http-client (visitado 10-07-2021).

[43] José Maldonado. *Truffle, la mayor herramienta de desarrollo para Ethereum*. es. Ene. de 2021. URL: https://es.cointelegraph.com/explained/truffle-the-biggest-development-tool-for-ethereum (visitado 10-07-2021).

[44] Gal Margalit. *React hooks lifecycle*. Jul. de 2021. URL: https://github.com/Wavez/react-hooks-lifecycle (visitado 18-02-2021).

[45] Julio Marín. *Centralized vs Decentralized vs Distributed: a quick overview*. en. Ene. de 2019. URL: https://medium.com/@juliomacr/centralized-vs-decentralized-vs-distributed-a-quick-overview-1f3bd17b8468 (visitado 11-02-2021).

[46] *moment*. en. URL: https://www.npmjs.com/package/moment (visitado 10-07-2021).

[47] *Namecoin, un DNS basado en blockchain*. URL: https://www.namecoin.org/ (visitado 12-02-2021).

[48] *New React Developer Tools – React Blog*. es. URL: https://es.reactjs.org/blog/2015/09/02/new-react-developer-tools.html (visitado 15-07-2021).

[49] Node.js. *Node.js*. es. URL: https://nodejs.org/es/ (visitado 15-07-2021).

[50] *npm*. en. URL: https://www.npmjs.com/ (visitado 15-07-2021).

[51] *Peepeth*. URL: https://peepeth.com/t/Peepth (visitado 12-02-2021).

[52] *Premier Diagramming, Modeling Software & Tools*. en-US. URL: https://astah.net/ (visitado 22-02-2021).

[53] *Principios de gobernanza en la blockchain*. es-AR. Sep. de 2020. URL: https://101blockchains.com/es/principios-de-gobernanza-blockchain/ (visitado 07-02-2021).

[54] *Proceso y Roles de Scrum*. es-ES. URL: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html (visitado 14-02-2021).

[55] *React Hooks, saca todo el potencial de React sin escribir clases*. es. URL: https://midu.dev/react-hooks-introduccion-saca-todo-el-potencial-sin-class/ (visitado 18-02-2021).

[56] *React Router: Declarative Routing for React*. en. URL: https://reacttraining.com/react-router (visitado 10-07-2021).

[57] *react-bootstrap*. en. URL: https://www.npmjs.com/package/react-bootstrap (visitado 10-07-2021).

[58] *Referencia de la API de los Hooks – React*. es. URL: https://es.reactjs.org/docs/hooks-reference.html (visitado 18-02-2021).

[59] Leomaris Reyes. *Aplicando el patrón de diseño MVVM*. en. Dic. de 2018. URL: https://medium.com/@reyes.leomaris/aplicando-el-patr%C3%B3n-de-dise%C3%B1o-mvvm-d4156e51bbe5 (visitado 25-05-2021).

[60] *Roles y Responsabilidades en un Proyecto Scrum*. es. URL: https://www.gestiondeproyectos-master.com/roles-y-responsabilidades-en-un-proyecto-scrum/ (visitado 14-02-2021).

[61] Toshendra Kumar Sharma. *Blockchain development platforms for blockchain apps*. en-US. Mayo de 2020. URL: https://www.blockchain-council.org/blockchain/top-2020-blockchain-platforms-for-building-blockchain-based-applications/ (visitado 13-02-2021).

[62] *Solidity*. en-US. URL: https://solidity-es.readthedocs.io/es/latest/ (visitado 18-02-2021).

[63] Sudeep Timalsina. *React Firebase Authentication with Context API*. en. Feb. de 2020. URL: https://medium.com/wesionary-team/react-firebase-authentication-with-context-api-a770975f33cf (visitado 15-07-2021).

[64] TresPuntos. *Atomic Design, última tendencia en diseño web*. es. Nov. de 2019. URL: https://trespuntoscomunicacion.es/tipsuxui/atomic-design-ultima-tendencia-en-diseno-web/ (visitado 11-07-2021).

[65] *Usando el Hook de efecto – React*. es. URL: https://es.reactjs.org/docs/hooks-effect.html (visitado 18-02-2021).

[66] *Usando el Hook de estado – React*. es. URL: https://es.reactjs.org/docs/hooks-state.html (visitado 18-02-2021).

[67] *web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation*. URL: https://web3js.readthedocs.io/en/v1.3.4/ (visitado 10-07-2021).

[68] Marcos Zocaro. *Blockchain para transparentar la Obra Pública*. es. Mayo de 2020. URL: https://marcoszocaro.com.ar/blockchain-para-transparentar-la-obra-publica/ (visitado 11-02-2021).