Universitat Oberta de Catalunya

uoc.edu

Aplicación descentralizada basada en sistema de incentivos

Nombre del estudiante: David Gómez Sancho
Plan de estudios: Máster Universitario en Ciberseguridad y Privacidad
Área del trabajo final: Sistemas de blockchain

Nombre Consultor: Alberto Ballesteros Rodríguez
Nombre Profesor responsable de la asignatura: Víctor García Font
Fecha de entrega: 01/06/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aplicación descentralizada basada en sistema de incentivos*
Nombre del autor:	*David Gómez Sancho*
Nombre del consultor/a:	*Alberto Ballesteros Rodríguez*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Sistemas de blockchain*
Idioma del trabajo:	*Español*
Palabras clave	*Blockchain, solidity, DApp, Ethereum, ERC20*

Resumen del Trabajo (máximo 250 palabras):

Creación de una Aplicación Descentralizada basada en un sistema de incentivos que se pueda aplicar en distintos ámbitos (Clubes, empresas, fundaciones...). Esta aplicación permitiría registrar actividades (asistencia a eventos, rendimiento de trabajo...), y en función de los resultados podría premiar con puntos o tokens (ERC20) a los integrantes de esta. La aplicación desarrollada se basa en la tecnología Ethereum y en la parte visual para que el usuario pueda interactuar se ha desarrollado una aplicación web. Por tanto, han sido requeridos conocimientos de blockchain y desarrollo de aplicaciones web adquiridos mientras se desarrollaba la aplicación.

Abstract (in English, 250 words or less):

Creation of a Decentralized Application based on an incentive system that can be applied in different areas (clubs, companies, foundations...). This application would allow to register activities (attendance to events, work performance...), and depending on the results could reward with points or tokens (ERC20) to the members of this. The developed application is based on Ethereum technology and a web application has been developed for the visual part so that the user can interact. Therefore, knowledge of blockchain and web application development acquired while developing the application has been required.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Se desea proporcionar un sistema basado en blockchain que permita registrar actividades y en función del desempeño de los miembros, se pueda premiar con incentivos en forma de token. Este sistema estará diseñado usando la red Ethereum.

Esto cubre la necesidad de aportar métricas y sistemas de incentivos positivos para la realización de diferentes actividades en diferentes ámbitos. Un ejemplo sería el de una empresa que desea implementar un sistema en el que se premie la asistencia a eventos de formación.

Respecto a las actividades deben servir para diferentes ámbitos, ya sea el desarrollo de tareas en el ámbito laboral o la asistencia a eventos como pueda ser de una fundación o una determinada formación.

El tipo de token utilizado será ERC20 que permitirá la concesión de los mismos a las diferentes actividades por parte de los usuarios que han asistido.

Asimismo, atendiendo a las consideraciones del tutor, se ha realizado una planificación mediante diagrama de Gantt que permite el uso del tiempo de una forma ágil, de manera que se empezó la memoria en las primeras etapas del proyecto, lo que permitió alargar el tiempo de desarrollo de la aplicación. Así se ha podido redactar la parte teórica de la memoria antes de estar acabada la aplicación y además se dispuso de mayor tiempo para perfeccionar el código.

A continuación se presentan las consideraciones y tomas de decisiones realizadas durante el desarrollo del proyecto.

1.2 Objetivos del Trabajo

- Entender los conceptos propios de blockchain.

- Entender la red Ethereum y su funcionamiento, así como sus limitaciones y puntos fuertes a la hora del desarrollo de aplicaciones descentralizadas.

- Obtener conocimientos en Contratos Inteligentes que permitan desarrollar la aplicación.

- Aprendizaje del lenguaje de programación Solidity.

- Investigar y obtener las tecnologías necesarias que puedan aportar valor a la aplicación a desarrollar.

- Crear la aplicación cumpliendo los estándares de calidad con un enfoque práctico.

- Eficiencia y calidad en el código entregado.

- Desarrollo del proyecto desde un enfoque que prima ante todo la seguridad informática.

1.3 Enfoque y método seguido

En primer lugar se empezó creando el estado del arte, es decir, investigando y dejando por escrito el estado de la tecnología actual que se vaya a utilizar en el proyecto. Seguidamente se investigó acerca de blockchain, Ethereum, contratos inteligentes y tecnología derivada de las mismas. A lo largo del desarrollo se han ido incorporando tecnologías que ofrecían una ventaja no tomada en consideración hasta ese momento.

La segunda fase consta del desarrollo de la aplicación. En ella se utilizaron los conocimientos teóricos obtenidos en la primera fase y se pusieron en práctica. Cada cierto tiempo, cuando se entendía que se había desarrollado una parte importante de la app o se tuvieron dudas acerca de qué dirección seguir, se consultó con el tutor si se estaba tomando una estrategia adecuada con el alcance del trabajo. Durante esta etapa se desarrolla la parte de la memoria que no necesita de la finalización de la aplicación, esto es el estado del arte y las cuestiones teóricas que se deben abordar antes del desarrollo.

Finalmente, tras el visto bueno del tutor y tras la finalización de la aplicación, se dio lugar a la fase de desarrollo de la memoria. Esta fase está exclusivamente centrada en el desarrollo de la memoria, concretamente en la parte de la memoria que necesita de la finalización y definición de la aplicación. En esta fase también es muy importante el feedback aportado por el tutor. Cuando se entrega la memoria se debe preparar la presentación del trabajo.

1.4 Planificación del Trabajo

- Planteamiento del problema y propuesta
- Definición alcance de aplicación
- Investigación tecnologías a utilizar
- Familiarización herramientas a utilizar
- Desarrollo de aplicación
    - Desarrollo Contrato inteligente
    - Desarrollo interfaz de usuario

- Análisis y prueba de la aplicación
- Redacción de la memoria de trabajo fin de máster
- Conclusiones de la propuesta
    - Preparación presentación
    - Defensa

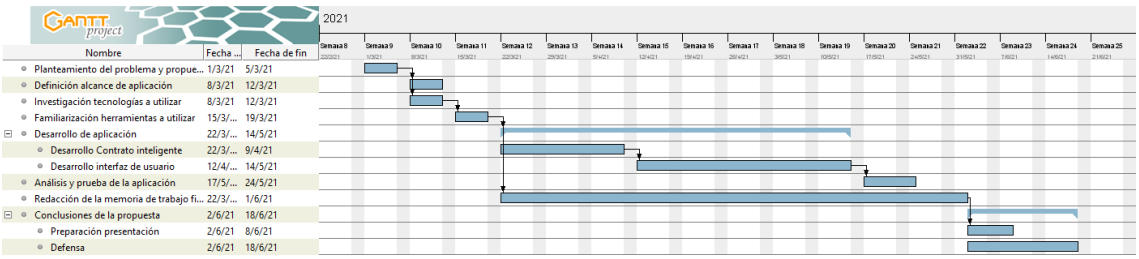

Figura 1.1: Diagrama de Gantt para la organización de las fases del proyecto.

1.5 Breve sumario de productos obtenidos

El trabajo ha consistido en el desarrollo de una aplicación descentralizada cuya lógica está basada en contratos inteligentes o Smart Contracts. Los contratos inteligentes consisten en una serie de normas y procedimientos que se programan para poder ser utilizadas en una red blockchain, en nuestro caso Ethereum. De esta forma los contratos inteligentes permiten llegar a acuerdos y realizar transacciones de forma segura y sin la necesidad de terceras partes involucradas. En la aplicación desarrollada en el presente trabajo se ha utilizado el estándar ERC20 para la creación y manejo de tokens que son utilizados activamente por el contrato inteligente.

Por otro lado se ha desarrollado una interfaz de usuario basada en tecnologías web para la interacción directa del usuario con el contrato inteligente. Esta parte está basada en las tecnologías html, css, javascript, React.js, Next.js y web3.js. A lo largo de la presente memoria se profundizará en cada una de las tecnologías expuestas, su funcionamiento y el valor que aportan a la aplicación.

1.6 Breve descripción de los otros capítulos de la memoria

En adelante se definirá el contenido capítulo por capítulo.

Capítulo 2: Estado del arte. Capítulo destinado a obtener un base de conocimiento disponible relacionada con el tema que se trata en la presente memoria. Para ello los primeros cuatro apartados explican de qué tratan las tecnologías fundamentalmente utilizadas desde un punto de vista teórico. El quinto apartado recopila una serie de artículos de investigación que se han consultado para estar al tanto de otras aplicaciones desarrolladas o datos de utilidad que han podido aportar. El apartado seis se destina a explicar

brevemente cada una de las tecnologías utilizadas desde una visión práctica, es decir todos los lenguajes de programación, librerías, frameworks y demás tecnología destinada a poder aprovechar el conocimiento teórico obtenido de los cuatro primeros apartados desde un punto de vista práctico. Finalmente el apartado siete explicará las herramientas y aplicaciones utilizadas que han permitido el desarrollo de la aplicación.

Capítulo 3: Arquitectura. Se trata del capítulo en el que se explica la estructura de la aplicación, sus componentes y la relación entre los mismos. A su vez se detallará cómo funciona el modelo de recompensas necesario para valorar las diferentes actividades y finalmente la conexión realizada entre la interfaz de usuario y la red Ethereum.

Capítulo 4: Análisis funcional. Detalla el funcionamiento de la aplicación atendiendo a criterios de funcionalidad, esto es, los requisitos que se pide a la aplicación para funcionar, qué tareas realiza y a qué funciones se debe llamar en la propia aplicación para cumplir las tareas que se requieran.

Capítulo 5: Conclusiones. Se expone un balance de los objetivos logrados. Asimismo se pone el foco en el futuro y las posibles expansiones de funcionalidad de la aplicación.

Capítulo 6: Glosario. Se ofrece una definición de cada uno de los términos utilizados en la memoria que son específicos del tema tratado y que es necesaria su explicación en un apartado aparte.

Capítulo 7: Bibliografía. Citas de la diferente bibliografía en la que se ha apoyado la presente memoria.

Capítulo 8: Anexos. Enlaces de interés que aportan información extra relacionada con el proyecto.

2. Estado del arte

A la hora de afrontar un proyecto resulta conveniente informarse todo lo posible sobre la materia relacionada que aporte un contexto que lleve al replanteamiento de la solución y a incorporar mejoras o recursos novedosos. En esta sección se atiende este requisito haciendo especial hincapié en aquellas soluciones que ayuden a profundizar en las tecnologías utilizadas en el proyecto. A continuación, se detallan los documentos de investigación que han resultado en una mayor aportación al proyecto.

2.1. Blockchain

Blockchain consiste en una estructura de datos compuesta de bloques interconectados entre sí, o transacciones, que hacen uso de mecanismos de cifrado y que se despliegan en una red p2p. Las bases conceptuales se definieron en un artículo que dio lugar al Bitcoin, desarrollado por una persona o grupo de personas bajo el pseudónimo de Satoshi Nakamoto [1].

La parte fundamental de blockchain es la estructura del bloque. El bloque posee un hash realizado a partir del uso de criptografía de los datos obtenidos del bloque anterior, permitiendo que si el bloque anterior cambiara toda la cadena de bloques quedaría invalidada. Esto es así porque si un bloque cambia también debe cambiar el hash creado a partir del mismo, por lo que si se encuentra un bloque corrupto el hash generado no coincidirá con el hash almacenado del bloque inmediatamente posterior. Otro dato almacenado en el bloque es una marca de tiempo que permite saber cuándo se llevó a cabo una determinada generación de bloque o minado. Finalmente el bloque contiene información sobre las transacciones. Esta estructura de bloque permite crear una cadena de bloques, debido a que los diferentes bloques apuntan al anterior mediante el hash, así hasta llegar al bloque raíz, y se asegura que la cadena en su conjunto es válida o no válida, debido a que en cuanto existe un hash que no coincida la cadena resulta invalidada. Esto provoca robustez para definir datos que han sido corrompidos.

Se llama minería de datos [2] al proceso de cálculo de hashes para una determinada cadena de bloques. Siempre que se realiza una nueva transacción en la red se debe volver a recalcular los hashes. Los responsables de creación y verificación de los bloques reciben el nombre de mineros, que obtienen una recompensa, ya sea Bitcoin, Ether o cualquier otra criptodivisa según la tecnología que se apoye, si son los primeros en realizar una determinada tarea.

Tal y como se verá en la sección dedicada a los artículos de investigación se ha consultado el artículo de *Computational Intelligence Approach for Municipal Council Elections Using Blockchain* [3] en el que se definen las principales ventajas de utilizar blockchain para sistemas de voto, blockchain mejora la seguridad y al ser distribuido evita que haya un cuello de botella que impida la disponibilidad. Otra ventaja es la de que es completamente trasparente y posee

mejor trazabilidad. Asimismo debido al comportamiento de la cadena de bloques se garantiza la integridad de los datos.

Particularmente estas son las características que señala de blockchain [3]:

- Inmutabilidad. Se refiere a que los datos en blockchain no pueden ser modificados o eliminados a priori, si se quiere cambiar un dato se deberá cambiar para todos los nodos de la red.

- Transparencia e integridad de datos. Todos los nodos pueden comprobar la transacción, esto es, cada uno puede saber si un nodo está corrupto o no, ya que deben llegar entre todos a un consenso.

- Persistencia. Los datos siempre son comprobados al firmar una transacción, si fallase se provocaría un fallo en dicha transacción.

- Anonimato. Referido a que no se debe saber quién ha enviado y recibido dinero en una transacción.

- Validación de datos. Para que no se pueda votar en caso de no cumplir los requisitos.

Debido a esto blockchain permite la creación de redes y aplicaciones confiables descentralizadas, sin necesidad de terceras partes de confianza. A lo largo del tiempo blockchain ha ido evolucionando, distinguiendo 3 versiones según se ampliaban sus usos y utilidad.

La red p2p en la que se basa blockchain consiste en una serie de nodos u ordenadores que almacenan cada uno una copia de cada transacción [4]. Esto permite que, en caso de haber un nodo con una cadena corrupta, la misma no se propague por la red y que el nodo que posea la copia no válida la deseche y obtenga la copia válida de los demás nodos de la red.

Fundamentalmente existen cuatro procesos en blockchain [3]:

- Algoritmo de consenso. Consiste en una serie de pasos a seguir por cada uno de los nodos, todos deben seguir el mismo comportamiento, por lo que el mayor número de nodos activos que utilice un mismo algoritmo impone su algoritmo a los demás, ya que la red en su conjunto debe seguir el mismo comportamiento.

- Contrato inteligente. Programa que permite trasladar el concepto de contrato al ámbito digital, pudiendo ser validado, mostrado y pudiendo ejecutar los términos del contrato o la negociación del mismo.

- Autenticación de datos. Envío de la identidad de un usuario a partir de un seudónimo.

- Firma digital. Se utiliza cifrado asimétrico para la validación de los datos. Cada usuario posee un par de claves, una pública y otra privada. Un

ejemplo de uso de las mismas sería cuando mediante la clave privada se firma un envío y al llegar al destinatario lo descifra usando la clave pública, asegurándose de este modo que la información la ha enviado el usuario que esperado.

Por lo tanto se considera la tecnología bockchain como base propicia para la creación de aplicaciones descentralizadas o Dapps, debido a sus características basadas en la no dependencia de una autoridad externa, criptografía que protege la integridad de la red y la redundancia de la información almacenada, que hace prácticamente imposible la propagación de inconsistencias y valores no válidos de una determinada cadena de bloques.

2.2. Ethereum

Ethereum consiste en una plataforma digital basada en blockchain [5] y que permite la creación y despliegue de contratos inteligentes. Fue definido por *Vitalik Buterin* en un artículo publicado en 2013 [6]. En el artículo se definen las bases de la tecnología, aunque con el paso del tiempo se ha modificado añadiendo nuevas características y puliendo las anteriores.

Ethereum al estar basado en la cadena de bloques consigue garantizar la integridad de los datos subidos a la red. Como se ha mencionado en el anterior apartado la forma en que funciona la cadena de bloques es la creación de hashes que permitan garantizar que un bloque de datos no ha sido modificado. Además el mismo bloque apunta a otro que almacena el valor del hash. De esta forma al estar enlazados los bloques cualquier cambio en uno de ellos invalida los siguientes bloques a los que se apunta. En una red como Ethereum lo que se intenta es provocar redundancias, ya que cada elemento/nodo de la red posee una copia de cada una de la cadena de bloques, por lo que si en algún nodo se modifica algún dato, ya sea por un error o por un elemento de carácter malicioso, se poseen las demás copias que coinciden con la versión correcta, permitiendo retirar la versión incorrecta.

La principal diferencia de Ethereum comparadas con tecnologías similares basadas en blockchain, como Bitcoin, es en la capacidad de programar contratos inteligentes y desplegarlos en la red. Mientras que Bitcoin está orientado únicamente a proporcionar un sistema de pagos y una divisa que no dependa de gobiernos y funcione de forma descentralizada, Ethereum consigue ir más allá y se orienta hacia la creación de una economía basada en normas impuestas por contratos, es decir se transfiere dinero, se realizan acciones, se obtiene información, en base a un programa, el cual permite un rango prácticamente ilimitado de posibilidades.

Asimismo se debe señalar que Ethereum es quasi-touring complete, esto es, que se puede realizar la programación de aplicaciones complejas que, en el caso de poseer recursos ilimitados, se podría realizar cualquier tarea o cálculo. En otras tecnologías como Bitcoin si bien se posee la capacidad de aprovechar scripts, no tienen la característica de ser quasi-touring complete [7] por lo que se encuentran limitados desde el punto de vista de generación de aplicaciones

descentralizadas. Para que se puedan generar aplicaciones basadas en la infraestructura Ethereum es necesario el concepto de gas, que es una medida para evitar un consumo ilimitado de recursos tal y como se verá más adelante en esta misma sección.

La criptomoneda nativa de Ethereum es el Ether, que tiene un valor convertible por cualquier otra divisa, ya pueda ser euros, dólares o bitcoin. Esta moneda funciona de forma muy similar a bitcoin, sistema descentralizado sin presencia gubernamental y apoyado en la cadena de bloques. De esta forma se pueden realizar pagos en la red Ethereum y se traspasan Ethers de una cuenta a otra. Aunque el Ether sea la divisa en la que se apoya, también se pueden crear tokens propios y, según dicte el contrato, se pueden intercambiar por Ethers o cualquier otra moneda. Sin embargo esto dependerá de las posibilidades que se programen en un contrato inteligente, es decir, se pueden crear tokens alternativos sin valor alguno más allá de representar un valor interno de una aplicación.

Conviene señalar que Ethereum se puede desplegar en redes privadas también, de hecho la aplicación desarrollada se despliega en Rinkeby, red orientada a pruebas de contratos inteligentes. Estas redes pueden crearse en cualquier ordenador o conjunto de nodos. En el artículo *Multi-service model for blockchain networks* se puede ver cómo se utiliza Ethereum para el desarrollo de una aplicación que aprovecha diferentes redes privadas basadas en Ethereum de forma concurrente [8].

Por otro lado conviene señalar que hay varios símbolos de unidades que representan el Ether pero en diferentes magnitudes. Por lo tanto cuando, por ejemplo, nos refiramos a Finney será el equivalente a 10^-3 Ethers. A continuación se muestra la lista de las unidades de Ethereum en la figura 2.1. [11].

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Figura 2.1: Unidades de medida de Ether.

La ventaja fundamental de Ethereum es la variedad de posibilidades que se pueden programar mediante contratos inteligentes. No obstante, esta capacidad de creación tiene un límite, las transacciones en Ethereum cuestan Ether, debido a que la minería y procesado de cadenas cuesta tiempo de

computación y por tanto recursos. Es en este punto en el que se introduce la idea de gas, que es una representación de una unidad de computación, es decir, nos permite medir el coste computacional de una función. Este coste computacional se traslada a coste de Ether en el campo GASPRICE, valor presente en todas las transacciones enviadas en Ethereum. Otro valor a tener en cuenta es el de GasLimit [9] que define un límite de gasto computacional y evita posibles ataques de denegación de servicio o posibles bucles infinitos que podrían consumir una cantidad prácticamente ilimitada de recursos [6].
El número de contratos inteligentes desplegados se encuentra creciendo día a día y se prevé que siga la tendencia, por lo tanto es una tecnología joven con futuro y potencial que debe ser explotada para encontrar nuevas soluciones a problemas de todo tipo.

Por último señalar que para el uso de Ethereum se necesita una dirección de cuenta, una clave privada y una clave pública [10]. La dirección de cuenta consiste en un identificador para poder realizar transacciones, enviar o recibir Ether o identificar al usuario que accede a la aplicación. La clave privada consta de 64 caracteres generados aleatoriamente y realiza la función de generar diferentes claves públicas que serán compartidas en función de las conexiones realizadas. Para la obtención de la dirección de cuenta se aplica el algoritmo de hash sha3 a la clave privada y se obtienen los últimos 20 bytes del resultado. La función principal de la clave privada es la de firmar cualquier transacción o mensaje que se envíe desde la cuenta Ethereum, mientras que la clave pública puede verificar que el mensaje enviado ha sido firmado por la cuenta que dice ser, de esta forma se consigue que las transacciones y mensajes enviados desde el usuario hasta la cuenta no puedan ser interceptados, modificados o realizados por cuentas ajenas. Estos mecanismos criptográficos de clave pública y privada se basan en tecnología de cifrado asimétrico ampliamente utilizada y de eficacia contrastada.

2.3. Smart Contract

Un Smart Contract o contrato inteligente consiste en una serie de normas o procedimientos a seguir para llevar a cabo una transacción. Esta transacción puede realizar cualquier función que se pueda programar, desde enviar tokens de una cuenta a otra siguiendo una serie de pautas hasta cálculos matemáticos o guardado de información. Un Smart Contract, al estar basado en blockchain, el código es transparente y si se encuentra desplegado se puede ver cada línea de código desarrollada. Esto ofrece una gran ventaja y es la de saber con exactitud las condiciones del contrato antes de ejecutarlo, es decir, cualquier persona puede someterlo a una auditoría y encontrar problemas en el código, pudiendo evitar vulnerabilidades o cualquier problema con el código. Sin embargo esto puede resultar en una consecuencia negativa, ya que la persona que demuestre un comportamiento no esperado del Smart Contract puede aprovecharse de la misma y no avisar al desarrollador para su corrección.

En el caso de Ethereum existen varios lenguajes que permiten la programación de los contratos para manejar su comportamiento, pero el más extendido es Solidity. Este lenguaje de programación está creado específicamente para

Ethereum, por lo que se encuentra limitado por sus características propias, pero también potenciado, ya que aprovecha toda la tecnología proporcionada y saca el mayor partido posible de ella. Asimismo Solidity está influido por los lenguajes de programación C++, Python y JavaScript [12]. Por lo tanto Solidity es un lenguaje potente que ofrece un amplio rango de herramientas para la creación de Smart Contracts y se ha utilizado para el desarrollo del presente proyecto.

2.4. DApp

Consiste en una aplicación basada en sistemas descentralizados [13], es decir, se ejecuta en redes descentralizadas p2p y aprovecha las ventajas proporcionadas de este tipo de sistemas. En el presente proyecto la aplicación DApp desarrollada se sostiene en la red descentralizada basada en Ethereum. Aunque existen otra serie de tecnologías en las que se pueden crear DApps, Ethereum es la más extendida y de las que posee mayor infraestructura que lo apoya.

Asimismo se puede considerar una DApp como una combinación entre un desarrollo de Smart Contracts, que actuaría como base de la aplicación, y una creación de una serie de interfaces de usuario destinada a la interacción del usuario con el Smart Contract desplegado en la red descentralizada. Para la interacción de la aplicación con la red descentralizada se suelen utilizar librerías que actúan de puente entre el Smart Contract desplegado y la interfaz de usuario. La librería utilizada en el presente proyecto ha sido Web3.js aunque existen otras posibilidades, como ethers.js [14] así como otras librerías para otros lenguajes que no sean JavaScript. En la aplicación se ha usado JavaScript, CSS, Html, React y Next.js para la realización de la interfaz de usuario o frontend mientras que para el desarrollo de contratos inteligentes se ha optado por Solidity. En la siguiente imagen se puede ver un esquema del funcionamiento de una DApp [15].

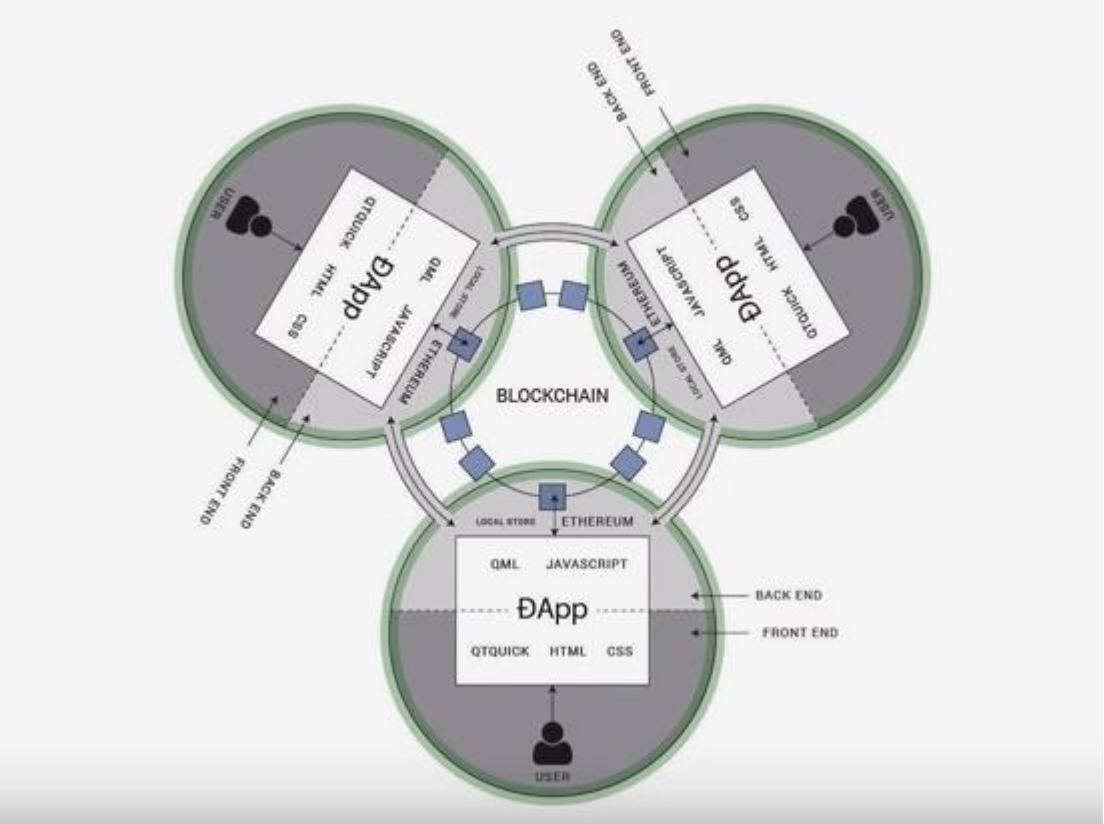

Figura 2.2: Esquema de funcionamiento de DApps.

Tal y como se verá en la siguiente sección se ha consultado el artículo *Computational Intelligence Approach for Municipal Council Elections Using Blockchain* [3] en el que resulta especialmente interesante la parte en la que se explica el desarrollo de una aplicación descentralizada y sus características. Tal como se ha desarrollado la aplicación del presente trabajo final, se ha utilizado Solidity como lenguaje base para la creación del contrato inteligente y se hace mención de tecnologías compartidas por la aplicación desarrollada y la del artículo, estas son, NPM, Truffle, Ganache y MetaMask. Sin embargo la principal diferencia es que la aplicación desarrollada en la presente memoria sí utiliza el sistema de tokens ERC20, mientras que la aplicación del artículo utiliza directamente Ether.

En definitiva el artículo resulta interesante para saber cómo se han desarrollado otras aplicaciones con sistema de votos, aunque la aplicación desarrollada no es exclusivamente un sistema de voto sí que posee una parte importante, específicamente la valoración de las actividades realizadas.

Otro artículo que se ha consultado *Easing DApp Interaction for Non-Blockchain Users from a Conceptual Modelling Approach* [16], ha servido para tener muy presente que la interacción por parte del usuario hacia el contrato inteligente puede llegar a ser muy complicada y se deben salvar conceptos que requieren una curva de aprendizaje pronunciada. Por ello lo ideal consiste en facilitar al usuario en todo lo posible la tarea de interactuar con la Dapp, intentando que la mayoría del proceso sea discreta y no requiera de demasiados conocimientos técnicos. Durante el desarrollo de la aplicación, principalmente la parte de interfaz de usuario, se ha creado una aplicación intuitiva que muestra en todo

momento el proceso a seguir y se da indicaciones al usuario para que se mantenga informado de lo que está ocurriendo, sin necesidad de profundizar en conceptos técnicos. El resultado es una aplicación descentralizada que aprovecha las ventajas de Ethereum y permite asegurar seguridad, integridad e interfaz intuitiva.

2.5. Artículos de investigación

A continuación se presentarán una serie de artículos de investigación que se han tenido en cuenta a la hora de la realización de la aplicación y han servido para estar al tanto de las últimas tendencias en el ámbito académico, así como para sentar las bases de la tecnología a utilizar.

Static Profiling and Optimization of Ethereum Smart Contracts Using Resource Analysis

En el presente artículo [17] se introduce de forma general el problema de consumición de gas en un contrato ethereum, lo que si no se tiene en cuenta puede dar lugar a sobrecostes que posteriormente repercutirán en el coste de ejecución de la aplicación. Este coste finalmente se traduce en que el usuario tenga que pagar una mayor cantidad de Ether. Según el artículo, se analizaron más de 40000 funciones de contratos inteligentes y se llegó a la conclusión que el 6.81% podían ser optimizados y el 4.19% podrían ser potencialmente vulnerables. Por lo tanto es muy importante tener en cuenta durante el desarrollo del contrato inteligente posibles métodos que permitan el ahorro de gas. Si el usuario puede ejecutar transacciones a bajo coste la aplicación tenderá a ser más accesible.

En este artículo se dan una serie de pautas que permiten ese ahorro de gas manejando correctamente la forma en que se procesan datos, variables y llamadas a funciones, es decir, se busca la eficiencia en el coste computacional que requiere la ejecución de una función en Ethereum.

Para conseguir el objetivo el artículo introduce su propia herramienta. Esta consiste en un analizador de funciones del contrato inteligente que define cuánto gas consume cada función, aportando una métrica de consumo a partir de Ethereum Virtual Machine, bytecode o el código fuente del contrato desarrollado en el lenguaje Solidity. Asimismo la herramienta permite llevar control de la memoria utilizada o el número de instrucciones ejecutadas.

La utilidad del análisis de gasto de gas viene reflejada en el propio documento, señalando que permite detectar cuándo una función excede del consumo de gas previsto, algunas veces dando lugar a un consumo desbocado, identificando ciertas vulnerabilidades ya conocidas y, al hacer un estudio de todos los posibles valores de entrada y como afecta al rendimiento de una función Ethereum, la detección de funciones que poseen demasiada variabilidad en el consumo de gas.

Finalmente señalar que el estudio ofrece consejos generales, como advertir que la mayoría de consumo de gas excesivo se produce en iteraciones de valores cuyo tamaño no están previamente definidos. Todas las consideraciones del artículo han sido aplicadas a la hora del desarrollo del contrato inteligente.

Computational Intelligence Approach for Municipal Council Elections Using Blockchain

En la línea de conocer otras aplicaciones desarrolladas previamente para la red Ethereum, el artículo [18] especifica cómo es una aplicación diseñada para su uso en elecciones, más concretamente en el ámbito local, de forma que el voto se realice de forma electrónica.

Los principales objetivos de la aplicación son identificar al usuario para evitar votos duplicados, garantizar la integridad del voto emitido, agilizar el proceso de voto disminuyendo el tiempo necesario para votar en comparación con sistemas no orientados a blockchain, aumentar la transparencia, garantizar la privacidad de los votantes en la medida de lo posible y ser capaz de la gestión responsable del sistema de votaciones. En la siguiente imagen se puede ver un esquema de funcionamiento de la aplicación [18].

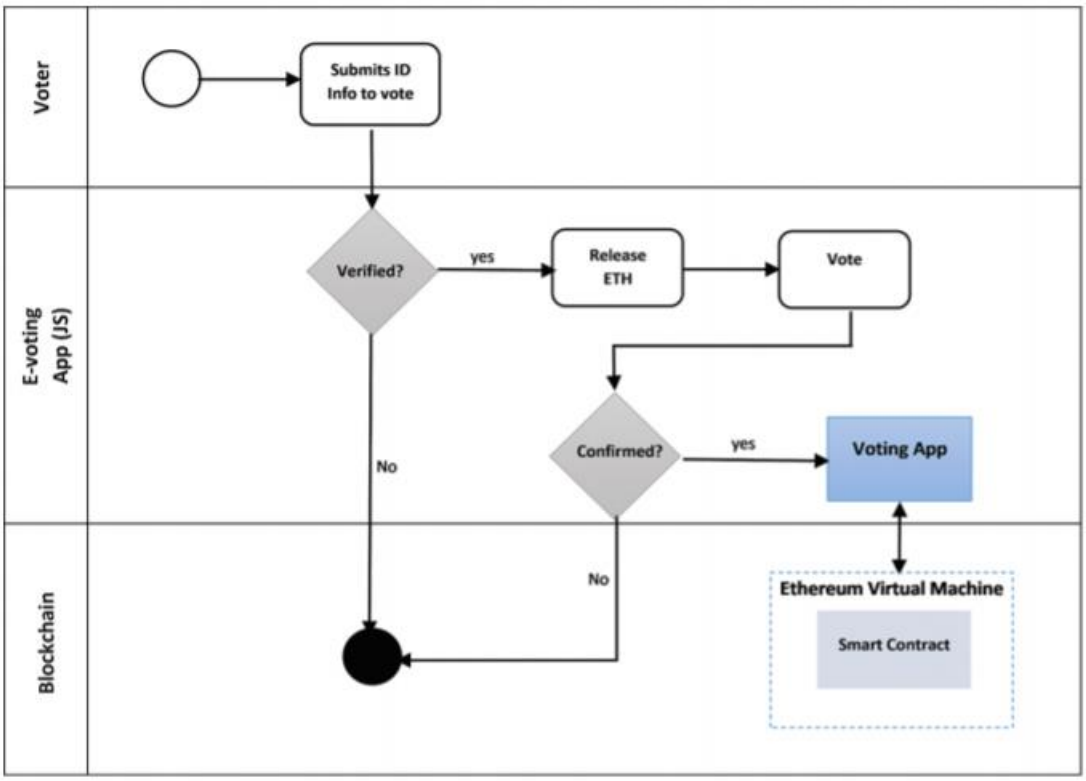

Figura 2.3: Diagrama de flujo de la aplicación de voto.

Multi-service model for blockchain networks

La visión del presente artículo [8] consiste en crear una red descentralizada que permita un servicio concurrente. Se busca esto para que la autoridad que regula los servicios sea descentralizada y de esta forma obtener precios por el servicio más justos. En este sentido se busca ofrecer una solución orientada a mejorar el procesamiento de transacciones para cualquier negocio que se apoye en Ethereum.

El artículo sirve para tener una toma de contacto con los conceptos clave de Ethereum, así como descubrir que existen opciones que se están creando para ofrecer, desde un punto de vista abstracto, nuevas posibilidades a los negocios en la red Ethereum, en este caso un modelo que permite el tratamiento concurrente de servicios.

An Organized Repository of Ethereum Smart Contracts Source Codes and Metrics

Cuando se pretende el desarrollo de una nueva aplicación lo primero que se debe hacer es revisar los programas que ya se encuentran finalizados para la misma tecnología que se va a utilizar. En este artículo [19] nos presentan un repositorio que permite buscar y encontrar diferentes contratos inteligentes.

Los problemas que se suelen encontrar al iniciar un desarrollo de un contrato inteligente recogidos en el artículo son los siguientes:

- El código y contrato desplegado a la red es inmutable, es decir, una vez que se sube a Ethereum ya no se puede actualizar o cambiar. Esto es un problema cuando surge alguna vulnerabilidad, al no poder actualizarlo se debe desplegar uno nuevo, pero habría que avisar a todos los usuarios del contrato de que el antiguo ya no es válido y que deberían usar el nuevo sin vulnerabilidades.

- La comunicación con la red blockchain se realiza mediante transacciones, por lo que la información solo se intercambia entre componentes internos de la propia red.

- La ocupación de memoria en blockchain generalmente tiene un coste, por lo que se debe tener en cuenta el tamaño del propio contrato desplegado y la eficiencia de las funciones desplegadas.

- Los desarrolladores no se pueden apoyar en soluciones ya probadas debido a la novedad de la tecnología utilizada, por lo que suelen cometer una y otra vez los mismos tipos de errores.

- Asimismo los desarrolladores no cuentan con medidas de calidad, complejidad y métricas en general que ayudan a aportar un código depurado y eficiente.

Por lo tanto para poder sortear dichos problemas se crea la propuesta de repositorio para obtener de forma fácil e intuitiva cualquier cantidad de código fuente y métricas de contratos inteligentes en Ethereum. Su propuesta es Smart Corpus y puede ser visitada en un enlace proporcionado en el artículo.

Smart Corpus fue desarrollada teniendo en cuenta la gran cantidad de contratos inteligentes desplegados utilizando tecnologías como base de datos orientada a documentos, lenguaje graph query y la plataforma de computación serverless.

Los procesos que sigue Smart Corpus para obtener el repositorio final son los siguientes:

- Data retrieving. Durante esta fase se obtienen los contratos inteligentes a partir de código fuente y application binary interfaces (ABIs) presentes en EtherScan. La información recolectada y depurada se almacena en el servidor de la aplicación.

- Data cleaning. Al desplegar un contrato inteligente se genera un identificador único. El problema surge cuando dos contratos con el mismo código fuente se despliegan varias veces, a efectos del identificador serán contratos diferentes, pero a efectos prácticos y de contenido son iguales. Durante esta fase se distingue ese tipo de casos y se evita la duplicidad de los contratos en la base de datos de la aplicación.

- Data modeling. Se analiza el contrato para obtener métricas y guardarlas en una base de datos orientada a documentos.

- Data querying. Almacenamiento de contratos inteligentes siguiendo una estructura propia en el servidor y almacenandolo en tres formas, código fuente, abi y bytecode. Los metadatos se guardan en una base de datos MongoDB.

En la siguiente imagen se puede ver un esquema por los pasos que sigue la aplicación según el artículo [19].

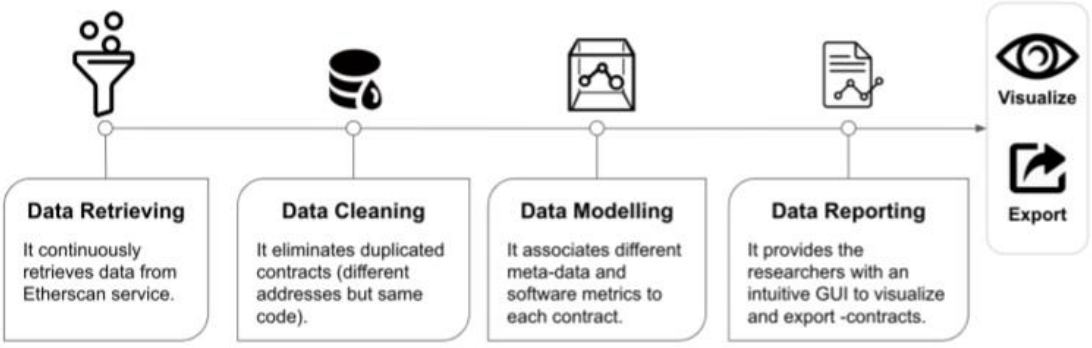

Figura 2.4: Esquema funcionamiento de la aplicación Smart Corpus.

El artículo ha servido para poder buscar otros contratos inteligentes sacando provecho de la herramienta desarrollada, y aunque la máxima versión soportada de Solidity es la 0.6 ha servido para hacerse una idea y aprender de otros contratos inteligentes desplegados y actualmente siendo utilizados en la red Ethereum.

Easing DApp Interaction for Non-Blockchain Users from a Conceptual Modelling Approach

Para el uso de una Dapp basada en blockchain por parte de usuarios, resulta necesario que los mismos posean una dirección blockchain, cartera y conocimientos sobre cómo realizar transacciones. La mayoría de personas no poseen estos conceptos que les permita usar una aplicación blockchain de forma intuitiva, por lo que resulta necesario introducir nuevas tecnologías y elementos que permitan suavizar la curva de aprendizaje u ocultar todo el proceso interno de uso de blockchain.
En el artículo [16] se propone una arquitectura software que resuelve los problemas antes mencionados. Para cumplirlo se apoyan en modelos UML y la creación de un middleware que actúa de puente entre la Dapp y los usuarios. Asimismo para demostrar el funcionamiento del middleware han desarrollado una Dapp.

2.6. Tecnologías utilizadas

En el presente capítulo se explicarán brevemente las tecnologías utilizadas para el desarrollo del proyecto.

JavaScript

El lenguaje Javascript fue creado por Brendan Eich en 1995 para aportar funcionalidad dinámica a las páginas web abiertas con el navegador Netscape [20]. Al principio solo ofrecía soporte de ejecución para la ejecución por parte del cliente y en entornos frontend pero con el tiempo y con la incorporación de Node.js, del que se hablará en capítulos posteriores, se pudo ejecutar en el lado del servidor y entornos backend.

Asimismo a lo largo de su historia ha tenido una gran cantidad de cambios, siendo el último más importante en 2015 cuando se lanzó la versión EcmaScript 6. En el presente proyecto ésta ha sido la versión escogida.

En el proyecto Javascript se ha utilizado para dotar funcionalidad a la parte frontend, esto es, actualizar datos, interactuar con el Smart Contract, gestión de botones y demás tareas que requieran de dinamismo en la vista. Para la gestión del Smart Contract también se ha utilizado JavaScript, es decir se encarga de su compilación y despliegue en Ethereum.

Html

Define la vista a nivel de elementos de la aplicación frontend. Html es un lenguaje de programación basado en etiquetas que se encarga de mostrar contenidos estáticos de una página web. En la aplicación se ha utilizado para mostrar las vistas, todo lo que se pueda ver sin funcionalidad está compuesto por Html.

Css

Lenguaje que sirve para definir los estilos de los elementos de la página web. Define por lo tanto formas, color, posición de los elementos, y cualquier característica visual de la aplicación. Conviene matizar que mientras Html define los elementos de una página web Css define las características visuales de estos elementos.

Solidity

Solidity es un lenguaje de programación destinado a la creación de Smart Contracts en Ethereum. SmartActivities, el contrato inteligente creado en el proyecto, se ha desarrollado utilizando el presente lenguaje. En el momento de empezar el proyecto la versión más reciente de Solidity era la 0.8.2 por lo que se decidió desarrollar el contrato inteligente en esa versión.

En el contrato inteligente SmartActivities se han utilizado diferentes tipos de estructuras propias de Solidity. A continuación se muestran cada una de ellas:

- Struct. Creada para ofrecer un tipo de dato que puede contener otros tipos de datos, y de esta forma se consigue tener una lógica similar al concepto de objeto de otros lenguajes de programación. Así, una estructura puede contener números, direcciones o cualquier otro tipo de estructuras admitidas. Gracias a esto se puede estructurar la lógica de cualquier tipo de dato, entendiendo la función de cada dato por el nombre asignado.

- Mapping. Similar a la estructura map de otros lenguajes. Se compone de valores de estructura tipo clave valor, esto es, se almacena información y se le asigna una clave única a esa información, de tal forma que cuando se desea acceder al valor o información solo se necesita la clave. Esta estructura resulta útil cuando se desea acceder rápidamente a un dato, sin necesidad de gastar recursos en acceder y comprobar los elementos de una lista uno por uno.

- Arrays. Se componen de un conjunto de elementos que se almacenan contiguamente unos de otros. Esta es la estructura elegida para almacenar las actividades.

Por lo tanto Solidity es un lenguaje de programación indispensable cuando se desea el desarrollo de Smart Contracts y específicamente diseñado para ello. Por último destacar la rapidez en la que se están efectuando cambios en el lenguaje, día a día se mejora, se pulen los detalles y se desarrollan nuevas funcionalidades.

React.js

Se trata de una librería basada en JavaScript diseñada para la creación de interfaces de usuario [21]. El uso de React.js mejora la rapidez y calidad en que se realizan la parte visual de las páginas web. Sus principales características son:

- Basado en componentes. Los componentes son partes iguales en las que se puede dividir una vista. Al estar basado en componentes se permite que se pueda reutilizar código.

- Declarativo. React.js está basado en cambios de estado, cada vez que se produce un cambio en una variable de estado la página web automáticamente cambia para actualizar al nuevo valor. Esto facilita la gestión de cambio de estado de cada uno de los elementos.

- Fácil de aprender. La curva de aprendizaje no es pronunciada por lo que la adaptación a la librería resulta relativamente rápida.

React.js es una tecnología ampliamente utilizada en el ámbito del desarrollo de aplicaciones web frontend debido a las características antes mencionadas, por lo que, debido fundamentalmente a su tratamiento de estados, se decidió utilizarlo en el proyecto. La versión utilizada corresponde a 17.0.2.

Next.js

Se trata de un framework basado en React.js que permite el manejo de servicios de servidor y la carga de páginas web en la parte frontend. En el proyecto se utiliza para obtener acceso a las diferentes páginas en función de su nombre, de tal forma que si se desea acceder a la vista que produce el archivo crearactividad.js únicamente se debe ir a la ruta “/crearactividad” y Next.js se encargará de cargarlo y mostrarlo.

Node.js

Consiste en un entorno de ejecución de aplicaciones backend utilizando el lenguaje Javascript [22]. En la aplicación se ha utilizado a la hora de compilar, testear y desplegar el contrato inteligente. La versión de Node.js empleada es la 16.1.0.

Npm

Es un gestor de paquetes creado para Node.js. Se encarga de la instalación, actualización y eliminación de los diferentes paquetes de terceros utilizado en la aplicación. También ofrece funcionalidad para ejecutar los paquetes, como por ejemplo a la hora de iniciar y ejecutar Next.js.

Web3.js

Web3.js es una librería que sirve de puente entre la parte frontend de la aplicación y el Smart Contract SmartActivities desplegado en Ethereum. También se utiliza a la hora de desplegar el contrato, ya que provee mecanismos para dicha gestión. La versión utilizada en la aplicación es 1.3.6.

Metamask

Se trata de un plugin para navegador que gestiona las cuentas Ethereum del usuario y permite la interacción con las mismas a través de Web3.js. Es indispensable ya que ayuda a que el usuario pueda manejar y comprobar qué transacciones está enviando a Ethereum y una estimación de su coste, así como ayuda a definir un límite de gas en cada una de las transacciones. Muestra un mensaje de confirmación al iniciar una transacción y al finalizar.

Por lo tanto es importante contar con Metamask si se quiere una experiencia completa con la aplicación y para tener un mayor control sobre la cuenta del usuario.

Infura API

Su web [23] permite la creación de un punto de entrada para la red Ethereum a través de su API. Cada vez que se despliega el contrato resulta necesario acceder a este punto de entrada.

Truffle

Se trata de un entorno de desarrollo y testing que adquiere de forma sencilla el proveedor que se utiliza luego en Web3.js. En la figura 2.5 se puede ver el momento exacto en el código en el que se utiliza para obtener el proveedor. El primer argumento es la frase mnemotécnica utilizada para tener acceso a la cuenta de usuario y la segunda es el punto de acceso para aprovechar la funcionalidad de Infura API.

```
const provider = new HDWalletProvider(
      "",
      "https://rinkeby.infura.io/v3/"
);
const web3 = new Web3(provider);
```

Figura 2.5: Obtención del proveedor en el código fuente de la aplicación.

Solc

Compilador de Solidity que se utiliza a la hora de compilar el contrato inteligente. Se pueden encontrar para varios tipos de plataformas, siendo la correspondiente a Node.js la que finalmente se ha utilizado.

Ganache

Librería que permite la creación de redes privadas para poder testear el contrato desde un punto más cercano a la realidad de la red Ethereum posible.

Mocha

Framework destinado a la facilitación de la tarea de testeo del Smart Contract. En el proyecto ha sido la tecnología de referencia para realizar pruebas.

2.7 Herramientas de trabajo

A continuación se muestran las herramientas que se han utilizado para llevar a cabo el proyecto.

Atom

Programa utilizado para la gestión, creación y edición de los archivos que se han desarrollado. Gracias a Atom se puede cambiar fácilmente entre archivos en un mismo directorio y ofrece una serie de ayudas visuales al resaltar el código en función del tipo de componente que se emplee. Además la creación y edición resultan en mayor rapidez y proporciona una función de sugerencias de nombres de variables o funciones desplegadas mientras se programa.

Remix

Es una aplicación disponible en línea que permite el desarrollo, despliegue y prueba de Smart Contracts desarrollados en código Solidity. Como primera toma de contacto para probar la funcionalidad de un contrato inteligente resulta

esencial. Además permite utilizar el lenguaje Yul, diferentes compiladores y también diferentes proveedores. Por último señalar que la aplicación tiene la función de debug, muy interesante si se quiere descubrir la razón del porqué no funciona la aplicación de la forma en principio esperada.

PowerShell

Se trata de un programa que permite ejecutar comandos en la consola de Windows. Ha sido de utilidad para ejecutar los programas anteriormente citados, crear los proyectos, añadir los módulos necesarios a la aplicación y compilar y desplegar el Smart Contract.

3. Arquitectura

En el presente capítulo se tratará de explicar la estructura de la aplicación poniendo especial énfasis en los elementos que la conforman y la relación entre los mismos. A continuación, se proporcionan punto por punto los elementos de los que se compone la aplicación.

3.1. Modelo de recompensas

La aplicación está basada en el intercambio de tokens entre diferentes cuentas Ethereum realizado a partir de las votaciones de actividades. Este intercambio se produce a través de tokens ERC20.

Un token ERC20 es una forma de representar activos basados en blockchain, que tienen valor intrínseco y que se pueden intercambiar entre diferentes cuentas [24]. Este tipo de tokens solo pueden ser emitidos en la red Ethereum. Los tokens ERC20 deben cumplir un modelo especifico de características comunes, ya que su aspiración es la de convertirse en un estándar.

Gracias a que se establece un estándar y un conjunto de normas común la compatibilidad aumenta si en el futuro se quisiera ampliar la funcionalidad y poder intercambiarse por otros tokens ERC20. Además el que se cumplan los mismos procedimientos en cada token facilita el desarrollo en futuros proyectos que utilicen este mismo tipo de token.

Las características principales de los tokens ERC20 son [25]:

- Poseen nombre y código de identificación. En la aplicación desarrollada para el presente trabajo el token ha recibido el nombre de “Stars” y el símbolo “STR”. Gracias a esta función los diferentes tokens son fáciles de identificar unos de otros.

- Posee una interfaz básica para manejar los aspectos de su emisión. Gracias a su interfaz es necesario únicamente una llamada al contrato inteligente para obtener datos como la cantidad total de tokens en circulación y la representación del token en diferentes formatos de precisión decimal.

- Se puede consultar fácilmente el saldo de cada una de las cuentas por parte de sus propietarios.

- La transferencia es sencilla debido a que posee el mecanismo nativamente.

- Proporciona un mecanismo de transferencia en base a aprobaciones del usuario, los tokens no se envían si no se aprueba previamente. Esto proporciona una capa de seguridad que impide las transferencias no legítimas.

En el caso de la aplicación desarrollada aprovecha la tecnología de los tokens ERC20 para crear el token Stars. Stars sirve para votar las diferentes actividades. El flujo de los tokens en la aplicación tendría los siguientes pasos:

- Al desplegar el contrato se generan 10^30 tokens Stars. Estos tokens se crean para la cuenta que despliega el contrato. Esto se ha programado de esta forma debido a que la función para crear nuevos tokens es sensible a ser atacada con fines maliciosos, por lo que conviene limitar su acceso únicamente a la dirección que ha creado el contrato.

- Los tokens Stars se distribuyen entre los diferentes usuarios a medida que el creador del contrato los añade. El nuevo usuario empieza con 1000 tokens, que se transfieren desde la cuenta del creador del contrato inteligente hasta la cuenta del nuevo usuario.

- Los usuarios, ahora que poseen tokens Stars, pueden votar en las actividades. Al votar en la actividad el usuario transfiere tokens Stars al usuario creador de la actividad. La cantidad de tokens Stars enviados depende de la puntuación dada por el usuario, si el usuario vota un cuatro, entonces cuatro tokens se enviarán a la cuenta del creador de la actividad.

- Esta transferencia provoca que el contrato inteligente actualice la puntuación de la actividad, añadiendo los tokens transferidos al creador de la actividad al resultado de puntuación de la actividad.

De esta forma se consigue que haya un intercambio de tokens entre los usuarios y, al ser la transferencia hacia los creadores de la actividad, se incentiva la creación de las diferentes actividades

3.2. Backend

Está formado por los componentes de la aplicación con los que el usuario no interactúa directamente, compone la funcionalidad última de la aplicación y se puede considerar su núcleo.

La parte backend de la presente aplicación está compuesta fundamentalmente por el contrato inteligente o Smart Contract. El Smart Contract compone la piedra angular de la aplicación. A partir de la funcionalidad del mismo parten los demás componentes.

A la hora de generar y desplegar el Smart Contract se ha llevado a cabo siguiendo la estructura mostrada en la figura 3.1, que consta de un directorio para almacenar el código fuente llamado “contracts”, un directorio para almacenar el código compilado, llamado “build” y el directorio principal compuesto por los scripts necesarios para compilar, “compile.js”, desplegar, “deploy.js” y realizar tests para el Smart Contract, “test.js”. Los demás componentes son los necesarios para que pueda funcionar Node.js, entorno de

ejecución basado en JavaScript tal y como se ha visto en el capítulo del estado del arte dedicado a las tecnologías.

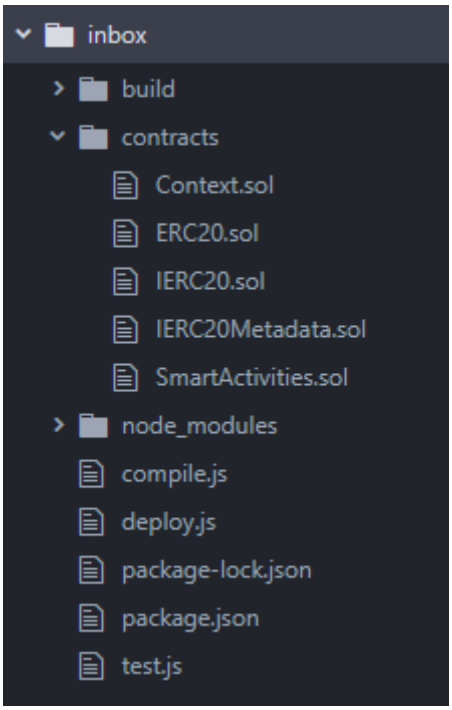

Figura 3.1: Estructura del proyecto en la parte backend.

Respecto a la estructura de los Smart Contracts se muestra en el esquema UML de la figura 3.2 que se puede observar a continuación, indicando cada uno de los componentes que definen la parte backend de la aplicación.

Figura 3.2: Diagrama UML de los Smart Contracts empleados en el proyecto.

En los siguientes capítulos detallaremos los componentes mostrados en el diagrama UML en mayor detalle, así como la infraestructura y scripts necesarios para compilar, desplegar y realizar test en los Smart Contracts.

3.2.1. Composición Smart Contract

La primera clase que aparece en el diagrama UML son las interfaces IERC20 y IERC20Metadata y la clase abstracta Context. Todas estas abstracciones tienen relación con la clase que los implementa o hereda ERC20. ERC20 es la clase encargada de crear y gestionar un determinado token de tipo ERC20. Se debe señalar que las clases para implementar el token han sido obtenidas del repositorio en Github OpenZeppelin [26], utilizando únicamente las clases necesarias para el funcionamiento de la clase ERC20. A continuación se explica cada una y su funcionalidad.

- IERC20. Es la interfaz empleada para implementar la clase ERC20, que crea y gestiona los tokens de tipo ERC20. En ella se indica que hay que implementar las funciones para gestionar la transferencia de tokens, así como la obtención del saldo y la cantidad total de tokens en circulación.

- IERC20Metadata. Consiste en una nueva interfaz que heredará la clase ERC20 y que consta de tres funciones que devuelven el nombre, el símbolo y los decimales del token.

- Context. Clase abstracta que proporciona datos necesarios para el funcionamiento interno del token en las transferencias.

La clase principal SmartActivities hereda de la clase ERC20 que permite la creación y manejo del token. Al crear una instancia de la clase SmartActivities se debe proporcionar el nombre y símbolo del token y con ello quedará creado. En el caso de la aplicación desarrollada al compilar SmartActivities se genera el token con los valores “Stars” y “STR”, es decir, el Smart Contract ya define el token a utilizar y la propia clase SmartActivities puede aprovechar todas las propiedades y funciones del token generado.

Al instanciar la clase también se debe tener en cuenta que es el momento en el que se general los 10^30 tokens totales que se usaran durante el tiempo de vida de la aplicación.

En SmartActivities se generan todas las funciones concernientes a la gestión de actividades, así como de los usuarios. Además se proporcionan una serie de estructuras internas que serán utilizadas para devolver o almacenar valores de variables.

3.2.2. Estructuras de datos

En el contrato inteligente SmartActivities se utilizan las siguientes estructuras:

- Activity. Estructura de Actividad. En ella se definen los valores que debe tener una actividad para su creación. Estos valores son activityId, title, resume, description, ownersName, addrOwner (para saber la dirección de la cuenta Ethereum que ha creado el contrato), startActivityDate, endActivityDate, startAllowedJoinActivityDate y endAllowedJoinActivityDate. Como puede verse se almacenan los datos relacionados a las descripciones de la actividad, así como fechas e identificadores de la propia actividad y el usuario que la crea.

- ActivityStatus. Estructura de estado de actividad. En ella se guardan los datos relacionados con la inscripción, asistencia y votación realizadas por cada usuario. Los campos son activityId, stars (número total de tokens Stars obtenidos por la actividad), votedCount(número de personas que han votado), inscribed, assisted y voted. Los tres últimos valores utilizan una estructura mapping, utilizado para evitar el consumo excesivo de recursos. Estos tres valores se utilizan para comprobar la asistencia, inscripción y voto de un determinado usuario. Se aprovecha la propiedad de Solidity en el que una estructura de tipo mappping siempre devuelve por defecto el valor false en caso de no haber sido asignado ningún valor. De esta forma si se quiere saber si alguien ha votado se busca el valor de la dirección de cuenta del usuario como clave y si es false significa que no se ha cambiado el valor y por tanto no ha votado, en caso de ser true se ha cambiado el valor, por lo que sí ha votado el usuario. Esto además permite evitar recorrer los elementos de una lista uno por uno desde el principio, evitando el gasto innecesario de gas.

- ReducedActivity. Estructura diseñada para devolver una versión reducida de la estructura Activity. En ella se devuelven los campos activityId, title, resume y endAllowedJoinActivityDate.

- CompleteActivity. Estructura diseñada para devolver todos los valores necesarios para poder mostrarlos al usuario, de esta forma no se deben hacer varias solicitudes a varias funciones para reunir los datos que no existen en la estructura Activity. Los campos son activityId, title, resume, description, ownersName, addrOwner, startActivityDate, endActivityDate, startAllowedJoinActivityDate, endAllowedJoinActivityDate, inscribed, assisted, voted, stars, votedCount y userStars.

- contractOwner. Dirección de la cuenta Ethereum del usuario que despliega el contrato inteligente. Se almacena para dar acceso a las funciones que solo están permitidas para el creador del contrato.

- Activities. Lista de actividades. Almacena todas las actividades creadas por los usuarios desde que el contrato inteligente se despliega.

- activitiesStatus. Estructura de estado para las actividades. En ella se indican los inscritos, asistencia, votaciones y número de Stars de la aplicación. La estructura es de tipo mapping por lo que si se tiene el id de actividad resulta sencillo y con bajo coste acceder al estado.

3.2.3. Funciones

Las funciones creadas en el Smart Contract de SmartActivities definen el comportamiento de la aplicación en sí, sin prestar atención al manejo del token, ya que las funciones que transfieren o crean tokens se llaman desde las propias funciones de SmartActivities. A continuación se exponen cada una de las funciones del contrato inteligente desarrollado:

- addActivity – Permite añadir actividad. Para ello será necesario proporcionar todos los datos y campos característicos de una actividad, mencionados anteriormente.

- getActivities – Permite obtener todas las actividades almacenadas en el contrato. Devuelve una lista de datos basado en la estructura Activity.

- getReducedActivities – Versión reducida del listado de actividades. Diseñado para reducir costes a la hora de mostrar la lista de actividades en la página principal. Devuelve una lista de datos basado en la estructura ReducedActivity.

- getCompleteActivity – Obtención de todos los datos de la actividad para un usuario, incluyendo datos de su estado y si el usuario que llama la función se ha inscrito, asistido y votado. Devuelve una serie de datos basados en la estructura ReducedActivity.

- getInscribed – Indica si el usuario se ha inscrito. En caso de respuesta afirmativa devuelve el valor booleano true, en caso contrario false.

- inscribeToActivity – Inscripción a la actividad por parte del usuario. Resulta necesario enviar como argumento el id de actividad. Se debe llamar dentro del plazo de fechas asignadas para la inscripción o el contrato inteligente devolverá un error.

- getAssisted – Indica si el usuario ha asistido a una determinada actividad. En caso de respuesta afirmativa devuelve el valor booleano true, en caso contrario false.

- assist – Marcado del usuario como asistido a la actividad. Para la ejecución de la función el usuario deberá haberse inscrito previamente y que la fecha de final de la actividad sea anterior a la fecha en la que se llama.

- getVoted – Indica si el usuario ha votado en una determinada actividad. En caso de respuesta afirmativa devuelve el valor booleano true, en caso contrario false.

- vote – Permite votar una actividad. Únicamente admitirá peticiones de usuarios previamente marcados como que han asistido a la actividad. Requiere el id de actividad y número de Stars a transferir y actualizará el marcador de la actividad, stars, y el número de personas que han votado en la actividad, votedCount.

- addUser – Para que el usuario reciba tokens y pueda usar la aplicación es necesario añadirlo previamente. Esta función solo puede ser llamada por el creador del contrato. La función pedirá el valor de dirección de cuenta del usuario al que añadir en el sistema.

3.2.4. Despliegue

Una vez desarrollado el código fuente del Smart Contract en un archivo .sol, se deberá compilar y desplegar. Para ello se utiliza el compilador solc disponible para Node.js.

El archivo utilizado para compilarlo es compile.js y puede ser encontrado en el directorio principal. En la figura 3.3 se muestra parte del código utilizado para compilar los contratos. Como puede observarse se compila la entrada, que corresponde al conjunto de contratos inteligentes, y se genera la variable output. Una vez obtenido el contrato compilado se crea el directorio en el que se guardarán los contratos, llamado “build”, se recorren cada uno de los contratos en la variable output y finalmente se guarda cada uno de ellos en ficheros con extensión .json.

```
var output = JSON.parse(solc.compile(JSON.stringify(input)));

console.log(output);

fs.ensureDirSync(buildPath);

for (let contract in output.contracts) {
  fs.outputJsonSync(
    path.resolve(buildPath, contract + ".json"),
    output.contracts[contract]
  );
}
```

Figura 3.3: Fragmento del archivo compile.js

Para ejecutar el script de compilación se accede con un terminal al directorio en el que se encuentra compile.js y se ejecuta el comando “node .\compile.js” tal y como se muestra en la figura 3.4.

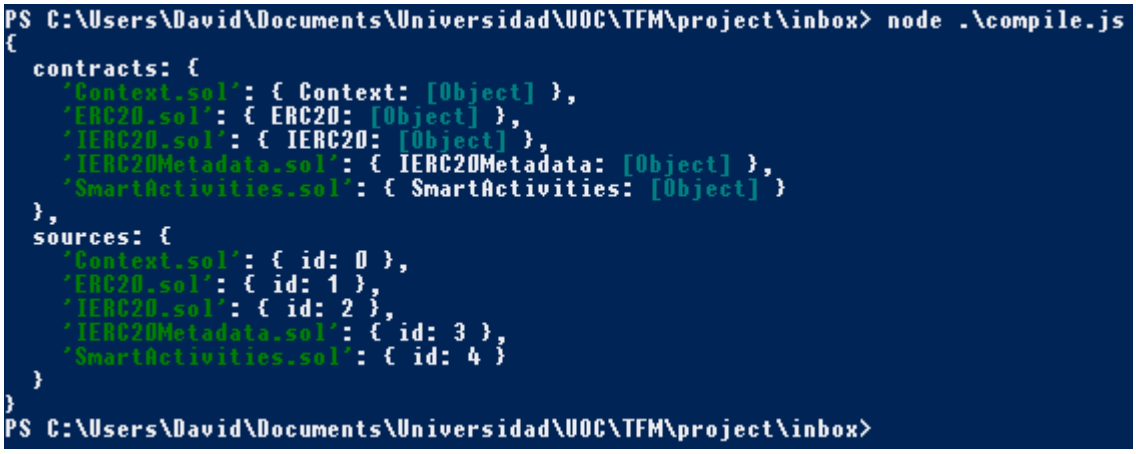

Figura 3.4: Resultado de ejecución del script compile.js

Una vez compilado y guardado el resultado en el directorio “build” se procede al despliegue del contrato. En este caso se ha creado el script deploy.js, disponible también en el directorio principal de la aplicación backend, al que se ha llamado “inbox”. El script se puede ver en la figura 3.5.

```
let source = fs.readFileSync("./build/SmartActivities.sol.json",
"utf8");
let smartActivities = JSON.parse(source).SmartActivities;

const provider = new HDWalletProvider(
  "",

  "https://rinkeby.infura.io/v3/"
);
const web3 = new Web3(provider);

const deploy = async () => {
      const accounts = await web3.eth.getAccounts();
      var myContract = new web3.eth.Contract([smartActivities.abi]);

      console.log("Attempting to deploy from account", accounts[0]);
      myContract.deploy({
            data: smartActivities.evm.bytecode.object
      })
      .send({
            from: accounts[0],
            gas: 10000000
      })
      .then(function(newContractInstance){
            console.log("Contract deployed to",
                  newContractInstance.options.address)
      });
}
```

Figura 3.5: Fragmento de script deploy.js utilizado para desplegar el contrato inteligente.

Como puede verse en primer lugar se carga en memoria el Smart Contract SmartActivities. Seguidamente se obtiene el proveedor que es el encargado de ofrecer una puerta de entrada a la red Ethereum. En capítulos posteriores se hablará en mayor detalle de las dos tecnologías utilizadas para obtener el

proveedor, Truffle e Infura API. El siguiente paso consiste en obtener el contrato y la cuenta que se utilizará para desplegarlo. Finalmente se despliega enviando el bytecode generado a la red Ethereum, en este caso la red de prueba Rinkeby, y en el momento que se reciba respuesta se imprimirá un mensaje de éxito por pantalla. Se debe tener en cuenta que siempre se debe indicar el gas límite tal y como se muestra en la figura 3.5.

3.2.5. Test

Para asegurarse que el contrato funciona se han hecho una serie de tests. Para ello han sido necesarias las tecnologías Ganache y Mocha. El script utilizado para llevar a cabo esta tarea ha sido test.js también disponible en la carpeta principal de la aplicación backend.

Las funcionalidades que han sido testeadas han servido para comprobar que el Smart Contract realizaba las comprobaciones pertinentes, por ejemplo, no admitir que usuarios se inscriban en actividades si no cumplen el rango de fechas habilitado para ello. También se han hecho tests para asegurar el funcionamiento correcto de cada función, si devuelve datos correctos o si cambian de estado correctamente las variables una vez el usuario se inscribe, asiste o vota en actividades.

3.3. Frontend

La parte de aplicación destinada a interactuar con el usuario se crea y define en la parte frontend. La aplicación posee una infraestructura web que permite interactuar directamente con el Smart Contract desarrollado, SmartActivities.

En la figura 3.6, que se muestra a continuación, se puede observar la estructura de directorios y ficheros de la aplicación frontend.

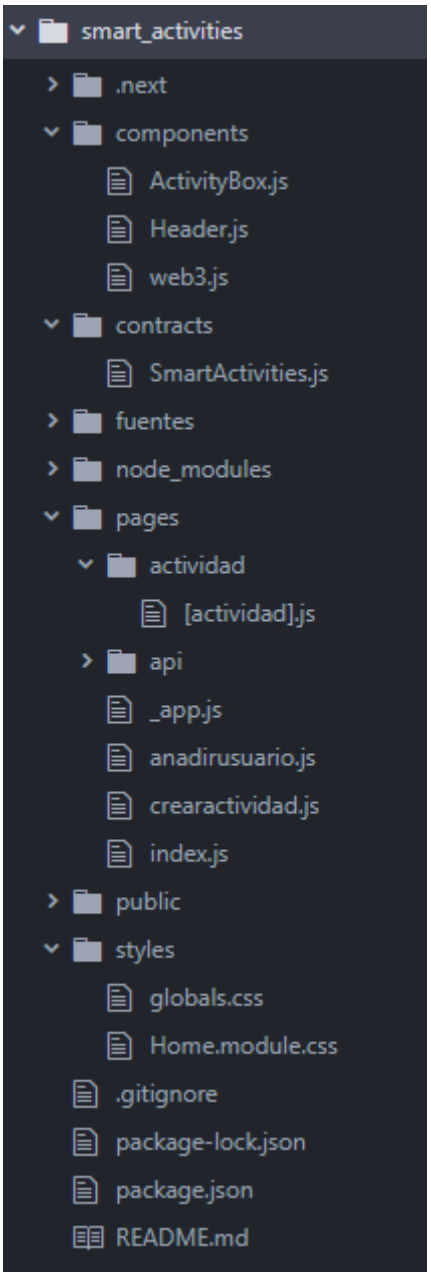

Figura 3.6: Estructura de la parte frontend de la aplicación.

En los siguientes capítulos se ahondará en más detalle los componentes que definen la interfaz de usuario y se explicarán las decisiones tomadas de cara a crear una aplicación intuitiva.

3.3.1. Components

En la aplicación hay tres componentes: ActivityBox.js, Header.js y web3.js.

En el caso de los dos primeros cumplen una función estética, en el que diferentes estructuras se repiten en toda la aplicación y se consigue reutilizar código transformándolos en componentes. La tercera, web3.js, se encarga de interactuar con la librería web3.js.

ActivityBox.js se utiliza para mostrar las cajas en las que se muestran las actividades en formato reducido en la página principal de la aplicación. La apariencia que muestran se puede observar en la figura 3.7.

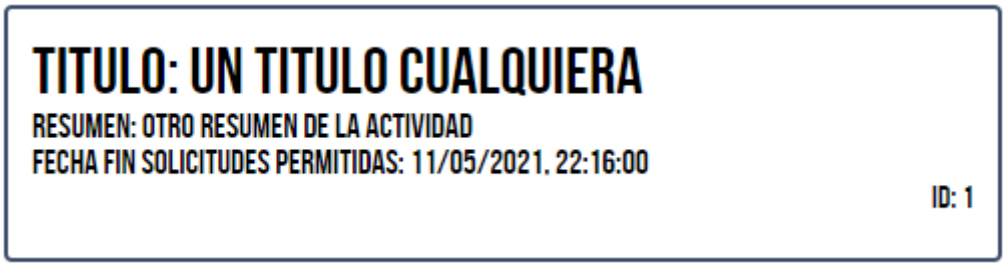

Figura 3.7: Representación del componente ActivityBox.js.

Por otro lado el componente Header.js representa la barra de navegación que se muestra en todas las vistas de la aplicación. Su representación se puede ver en la figura 3.8. Como se ha comentado anteriormente posee enlaces para acceder a la página principal, parte de la vista en la que se lee "SMARTACTIVITIES", y enlaces para crear actividad y añadir usuario. La ventaja del componente es que siempre será el mismo en toda la aplicación, por lo que el usuario se podrá familiarizar rápidamente con la barra y se reutilizará código de forma eficiente.

Figura 3.8: Barra de navegación construida a partir del componente Header.js.

Finalmente el componente web3.js utiliza la librería de igual nombre para utilizar como proveedor Metamask en caso de estar disponible y si no el proveedor será Infura API. En cualquier caso siempre se recomienda utilizar Metamask, ya que se consigue un manejo más intuitivo de las cuentas Ethereum. El componente se exporta ya que lo que se pretende es instanciar el objeto web3 que actúa de puente entre la aplicación y la red Ethereum. A la hora de acceder a cualquier función o dato del Smart Contract se deberá llamar al componente web3.

3.3.2. Contracts

En el directorio contracts se almacenan las interfaces necesarias para poder acceder al contrato en Ethereum junto con la dirección del contrato en la red. En las interfaces se definen las funciones y las normas que se deben seguir para acceder a ellas, así como características como los datos devueltos y los nombres de las funciones.

Únicamente se tiene un Smart Contract con el que se debe interactuar. Este contrato es SmartActivities y es el contrato que se exporta en SmartActivities.js y que será utilizado en la aplicación.

3.3.3. Pages

Las pages contienen el código necesario para implementar las vistas de la aplicación. A continuación se verán uno por uno los elementos pages utilizados en la aplicación frontend.

_app.js

La primera página que se puede ver es _app.js y es utilizada para cargar en la aplicación los componentes, así como los estilos CSS. Su estructura es simple tal como puede verse en la figura 3.9 y es un componente fundamental para que la aplicación se pueda ejecutar debido a que corre bajo el framework Next.js.

```
import '../styles/globals.css'

function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />
}

export default MyApp
```

Figura 3.9: Componente _app.js.

Página principal

La página principal corresponde a index.js, en ella se muestran todas las actividades en formato reducido tal y como se puede observar en la figura 3.10.

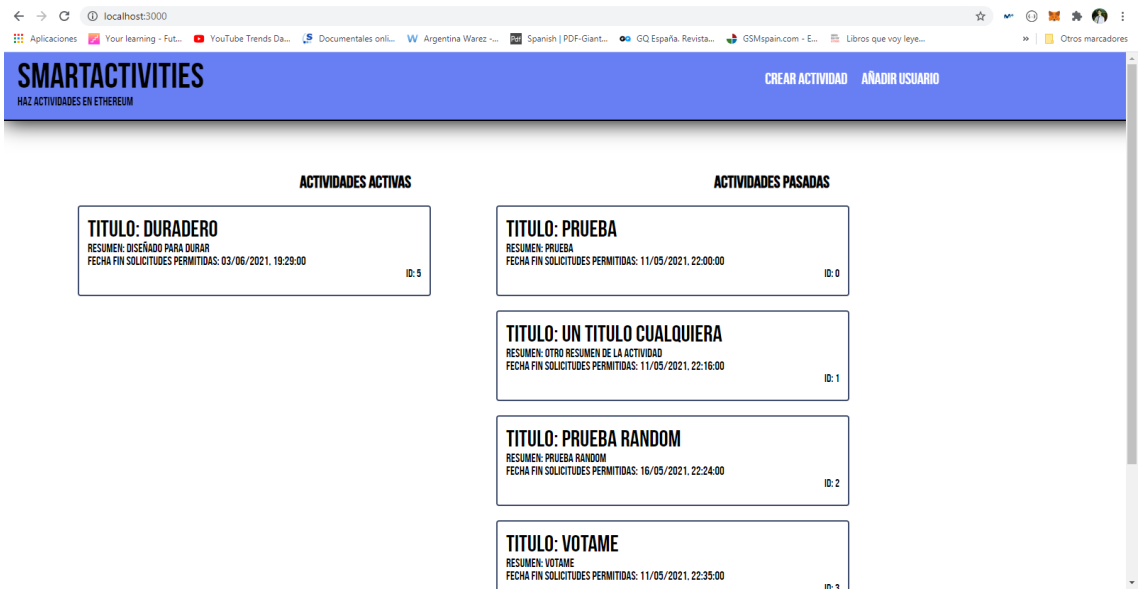

Figura 3.10: Vista principal de la aplicación

Como se ha comentado anteriormente la barra de navegación y las cajas se reutilizan en la vista. Si se pulsa en cualquiera de las cajas se accederá al detalle de la actividad.

Detalle de actividad

El detalle de actividad viene definido en [actividad].js dentro del directorio actividad, y en ella se representa la dinámica de inscripción, asistencia y voto de la aplicación. En la figura 3.11 se muestra la vista habitual de este componente.

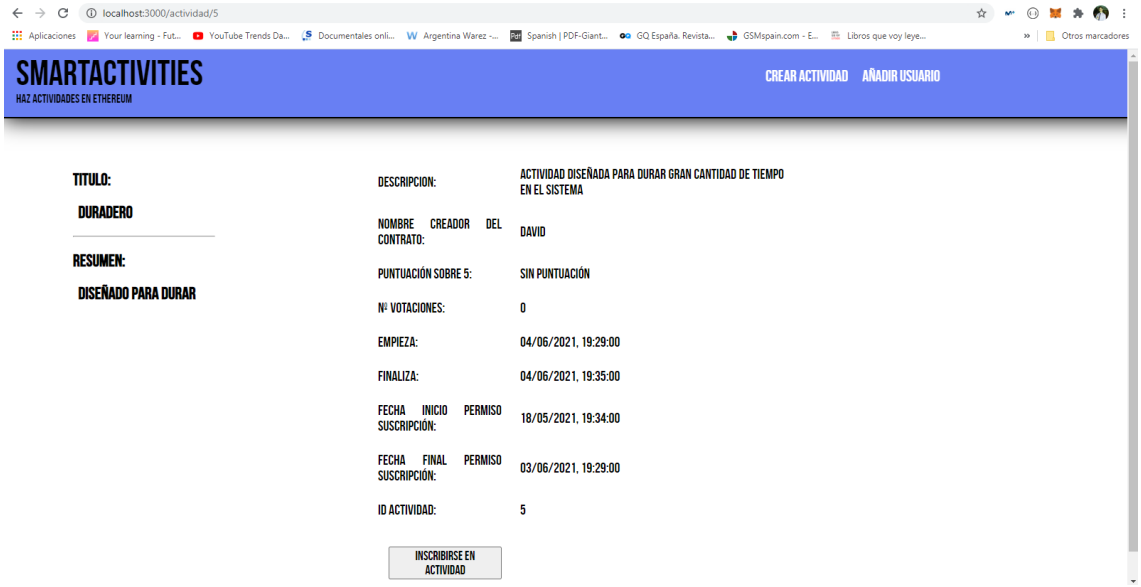

Figura 3.11: Vista de detalle de actividad.

El elemento clave de la vista es el botón, ya que permite realizar las tareas antes mencionadas y cambia en función del estado de la actividad, es decir, si el usuario se ha inscrito a la actividad mostrará un botón diferente que cuando el usuario ha sido marcado como asistido. Los diferentes botones de la presente vista se muestran en la figura 3.12.

Figura 3.12: Diferentes botones en la vista de detalle de actividad.

El primer botón se muestra cuando el usuario no se ha inscrito y está abierto el plazo para la solicitud de inscripción. El segundo botón se muestra cuando el usuario se ha inscrito y la actividad ha finalizado. El tercer botón se muestra al confirmar el usuario que ha asistido a la actividad, permitiéndole de este modo votar. Por último el cuarto botón se muestra cuando una transacción se está llevando a cabo en la red Ethereum. Cuando no se cumple ninguna de las condiciones anteriormente mencionadas no se muestra ningún botón.

Asimismo cada vez que se procese una transacción satisfactoriamente en la red Ethereum se mostrará el mensaje de la figura 3.13.

Figura 3.13: Mensaje informativo de transacción exitosa.

Añadir usuario

El archivo proporciona la vista para añadir un usuario para que pueda votar. Se debe proporcionar la dirección de la cuenta de usuario en Ethereum que se quiere añadir al sistema para que se transfieran 1000 tokens Stars. Esta vista está específicamente diseñada para la cuenta que despliega el contrato, las demás cuentas, aunque intenten realizar la transacción, no podrán debido a que el Smart Contract rechazará la transacción. En la figura 3.14 se puede ver la vista final de la página.

Figura 3.14: Vista de la página añadir usuario.

Crear actividad

La vista que corresponde a crear actividad es la que proporciona el archivo crearactividad.js. En la figura 3.15 se muestra la interfaz resultante.

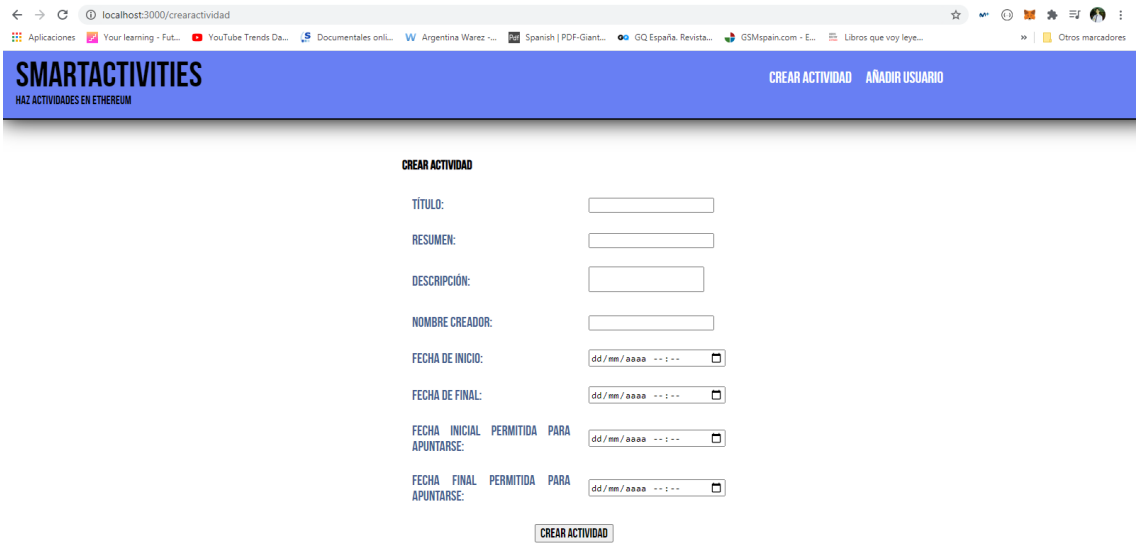

Figura 3.15: Vista para crear actividad.

Como puede verse consta de un formulario en el que se obtienen los datos de la actividad y el botón para enviar la transacción al Smart Contract y crear la actividad.

Es muy importante que los campos cumplan las normas de que las fechas no se solapen entre sí y que no haya inconsistencias como que la fecha de inicio sea posterior a la fecha final. Si no se cumplen estos requisitos la transacción no tendrá éxito y no se creará la actividad.

3.3.4. Styles

En el directorio styles se definen los archivos CSS. Estos archivos definen la apariencia de los elementos que se muestran. Esto es definir sombras, colores, la fuente de los elementos escritos o cualquier otro elemento visual de cualquier componente en la aplicación. También se define la posición en la pantalla de cada elemento de la vista. Por tanto es una sección dedicada a programar y dotar de estilo los elementos visuales de la aplicación.

3.4. Conexión Smart Contract

Para realizar la conexión se utiliza el componente web3.js. La forma en la que se utiliza es en primer lugar instanciando un objeto de la librería web3.js. Para generar la instancia se necesitará un proveedor que será por defecto Metamask o infuraAPI en caso de no poseer el usuario Metamask. En la figura 3.16 se muestra cómo se utilizan los componentes.

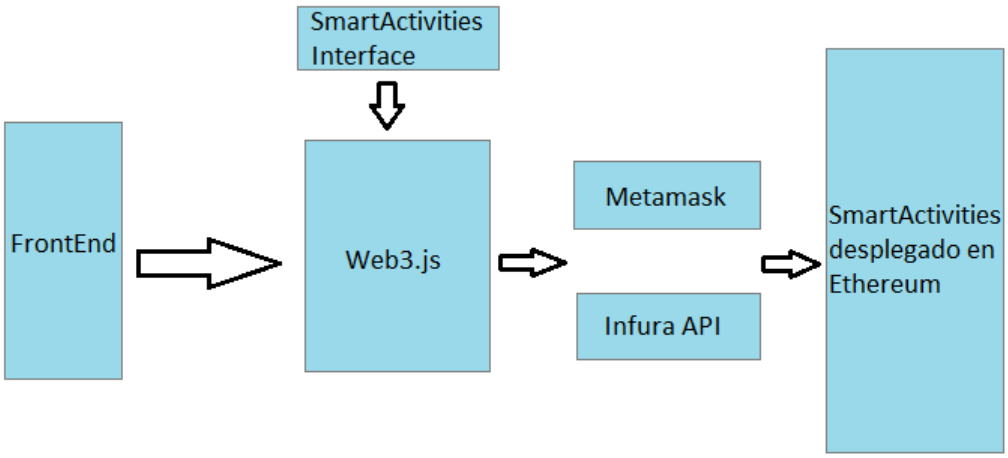

Figura 3.16: Esquema que muestra los diferentes componentes utilizados para acceder al Smart Contract desde el frontend de la aplicación.

Como puede verse el frontend siempre interactúa con web3.js cuando se desea llamar a cualquier función del contrato inteligente desplegado. Al instanciar Web3.js se obtiene por un lado el contrato a partir de la interfaz de

SmartActivities que se puede encontrar en el directorio contracts, y por otro lado define el proveedor que puede ser o Metamask o Infura API. A partir de los dos datos se puede acceder a SmartActivities desplegado en la red Ethereum. SmartActivities interface posee la especificación de cada uno de los métodos del contrato, la forma de acceder a ellos, así como la dirección del contrato desplegado en Ethereum. En la figura 3.17 se puede ver un ejemplo de llamada a función del Smart Contract a través de web3.js.

```
const activity = await
      smartActivities.methods.getCompleteActivity(
            this.props.actividadId).call();
```

Figura 3.17: Llamada en código a SmartActivities.

4. Análisis funcional

En el presente capítulo se detallará la aplicación desde un punto de vista completamente funcional, detallando qué tareas se espera que se cubran y el comportamiento general de la aplicación.

4.1. Despliegue del contrato

Para que la aplicación pueda funcionar necesitará de un contrato inteligente o Smart Contract desplegado en la red Ethereum. El primer paso, por tanto, consiste en la compilación y despliegue del contrato. Este proceso se ha abordado en mayor detalle, cómo se lleva a cabo y los pasos seguidos para lograrlo, en el capítulo 3 arquitectura de la aplicación.

El Smart Contract se ocupará del funcionamiento backend de la aplicación, es decir, todas las operaciones que el usuario no puede observar ni se le muestran pero que ocurren en la red Ethereum. Al desplegar el contrato se obtendrá una dirección de bloque de Ethereum que permite enlazar con la aplicación frontend que se encarga de servir de puente entre el usuario y Ethereum.

La primera acción que se realiza al desplegarlo es la creación de 10^30 tokens transferidos al usuario que despliega el Smart Contract en Ethereum. Estos tokens, que en la aplicación reciben el nombre de Stars (STR), se irán repartiendo entre los usuarios a medida que vayan siendo añadidos.

Asimismo al crear el contrato se almacena la dirección de la cuenta Ethereum lo que más tarde permitirá que se ejecuten funciones que solo puede realizar el creador del contrato.

4.2. Añadir usuario

Para el funcionamiento de la aplicación será necesario que el usuario que desee utilizar la aplicación posea tokens Stars. Este tipo de token creado específicamente para uso interno de la aplicación sirve para poder valorar las actividades en las que el usuario se ha inscrito. El usuario posee inicialmente 1000 tokens Stars y transfiere los tokens a las cuentas de usuario que crean actividades a medida que se votan actividades.

Para añadir el usuario se habilita una función llamada addUser en el Smart Contract. Esta función solo puede ser llamada por parte del creador del contrato, en caso de que se llame por cualquier otro usuario lanzaría un error. Para poder aprovechar esta función se habilita un apartado de la página web desarrollada en la ruta “/anadirusuario”. En la página se muestra un formulario que pedirá la dirección de la cuenta del usuario que se desea añadir a la aplicación y un botón en el que se da la opción de añadir usuario, tal y como puede verse en la figura 3.14 en el anterior capítulo.

4.3. Visualización de resumen de actividades

La página principal mostrada al usuario será la que corresponde al resumen de todas las actividades almacenadas en el Smart Contract. En ella se mostrará un resumen de cada una de ellas, siguiendo una interfaz intuitiva en el que cada actividad es incluida dentro de un elemento de tipo caja. En la misma se muestra el título, el resumen, la fecha de vencimiento para solicitar la inscripción a la actividad y el id de la actividad. Cada una de las actividades que se muestran en las cajas conducen a la página de detalle de la actividad si se pulsa sobre ellas.

Las actividades serán mostradas en dos grupos, actividades activas y actividades pasadas. El apartado de actividades activas mostrará las actividades almacenadas en el Smart Contract en las que todavía se puede inscribir el usuario, es decir, la fecha de vencimiento para inscribirse no ha sido superada en el día de acceso a la aplicación. Este apartado se encontrará localizado en la parte izquierda de la página principal. Por otro lado el apartado de actividades pasadas muestra todas las actividades en las que el usuario ya no tiene opción de inscribirse o que se ha inscrito el usuario pero ya no se admiten solicitudes para inscribirse por parte de otros usuarios. Este apartado tiene la función de llevar el registro de las actividades que se han producido y saber cuánta puntuación y tokens Stars han obtenido cada una de ellas. El apartado se localizará en la parte derecha de la página principal.

Todas las páginas de la aplicación frontend constan de una cabecera o barra de navegación. En ella se puede encontrar un enlace que redirige a la página principal, así como el enlace “crear actividad”, que muestra la página para poder crear una actividad nueva en el sistema, y por último el enlace “añadir usuario”, que muestra la página antes mencionada que ofrece la posibilidad de añadir un nuevo usuario para que pueda aprovechar la funcionalidad de voto en el sistema.

La función que se llama en el Smart Contract corresponde a getReducedActivities, que proporciona la lista con cada actividad y los datos que se mostrarán en la página principal. La ventaja de la función es que solo se muestran los datos imprescindibles y eso revierte en una reducción de uso de recursos por parte del Smart Contract.

El objetivo de la página principal es el de servir como toma de contacto y ventana para poder acceder a la interfaz general de la aplicación y dar la opción al usuario de navegar entre las diferentes actividades disponibles en la aplicación. La interfaz de usuario correspondiente a la presente funcionalidad puede observarse en la figura 3.10.

4.4. Visualización completa de actividad

Una vez se ha pulsado en la actividad en la página principal se muestra el detalle de actividad. En la presente vista se pueden observar todos los datos guardados de la actividad en el Smart Contract.

Estos datos son título, resumen, descripción, nombre de creador del contrato, puntuación, número de votaciones, fecha de inicio en el que se lleva a cabo la actividad, fecha final de la actividad, fecha inicio desde la que se admiten inscripciones, fecha final hasta la que se admiten inscripciones e id de la actividad.

La ruta para llegar a la página que corresponde a la actividad es “/actividad/{id}” siendo “{id}” el identificador de la actividad.

La función a la que se llama en el Smart Contract para obtener la información es getCompleteActivity. La función necesitará el id de la actividad como argumento y a partir del mismo devolverá toda la información relacionada con la actividad que se encuentra guardada en Ethereum.

Por último en la parte inferior de la página se encontrará un botón cuyo comportamiento dependerá del estado de la actividad teniendo en cuenta cada usuario. En los siguientes apartados se ve el comportamiento del botón mostrado.

La vista asociada a la función de visualización completa de la actividad se corresponde a la figura 3.11., previamente mostrada.

4.5. Inscripción a actividad

Una vez creada una actividad se precisa un rango de fechas en las que se admitirán inscripciones a la misma. En este rango de fechas en la página de detalle de actividad se mostrará un botón en la parte inferior en el que se puede leer “inscribirse a la actividad”. El botón no se muestra si no se encuentra en el rango de fechas válidas para inscribirse. Asimismo una vez inscrito el usuario en la actividad no se vuelve a mostrar el botón.

La información de si el usuario se ha inscrito o no nos llega a partir de la función getInscribed del Smart Contract, que nos proporciona un valor verdadero o falso.

Si el usuario pulsa sobre el botón la aplicación front llamará a la función inscribeToActivity del Smart Contract con el argumento id de actividad. El Smart Contract tardará un tiempo en completar la transacción, momento en el cual se desactiva el botón y se mostrará el mensaje “procesando...”.

Una vez completada la transacción correctamente se eliminará el botón y se mostrará el mensaje "Operación realizada con éxito" en el lugar en el que

anteriormente se encontraba el botón. El usuario se habrá inscrito correctamente en la actividad.

4.6. Asistencia a actividad

Cuando el usuario ya se ha inscrito a una actividad y ha asistido a la misma el usuario puede registrarse en el sistema como inscrito.

Para ello debe acceder a la página de detalle de actividad cuando la fecha de finalización de la actividad es anterior a la fecha en la que el usuario accede a la aplicación. En ese momento, si el usuario se ha inscrito anteriormente, se mostrará un botón en la parte inferior de la página con el texto “Pulse si ha asistido a la actividad”.

Cuando el usuario pulse sobre el botón la aplicación front llamará a la función assist del Smart Contract con el id de la actividad como argumento. De nuevo mientras se completa la transacción en Ethereum se mostrará al usuario el botón deshabilitado mostrando el texto “procesando...”. Una vez completado el proceso el botón será eliminado mostrando la vista que corresponde a la fase que permite votar la actividad.

4.7. Votar actividad

Al asistir a la actividad y actualizarlo en la aplicación el usuario podrá votar. En este paso es fundamental que el creador del contrato haya añadido al usuario en el sistema, si no le será imposible votar.

Si el usuario quiere votar deberá ir a la página de detalle de actividad. Una vez dentro se encontrará que en la parte inferior de la página hay un selector de rango, de cero a cinco, y un botón con el texto “votar”, tal y como se muestra en la figura 4.1.

Figura 4.1: Elemento visual empleado para votar la actividad.

Cuando el usuario seleccione la cantidad con la que desea votar y pulse el botón el frontend llamará a la función vote del Smart Contract, esta vez con el id de la actividad y el número comprendido entre el cero y el cinco. De nuevo se muestra el botón con texto “procesando...” y cuando la transacción se completa se muestra el mensaje de “Operación realizada con éxito”. Una vez se ha votado la información de número de votos y la media de la puntuación obtenida de los votos se actualiza.

En este punto resulta importante señalar que el número de tokens Stars con el que vota el usuario se trasladan desde la cuenta del usuario a la cuenta del

creador de la actividad y la actividad subirá de puntos igual al número de tokens transferidos. El número máximo de tokens a transferir es de 5.

4.8. Crear actividad

Cuando se desea crear una actividad el usuario debe acceder a la página con ruta “/crearactividad”. Se puede acceder a ella pulsando sobre el enlace disponible en la parte superior derecha de cualquier página de la aplicación, ya que la barra de navegación es igual entre todas las vistas, o también se puede acceder escribiéndolo directamente en la url en la barra del navegador web.

La página mostrará un formulario con los siguientes elementos: título, resumen, descripción, nombre del creador, fecha de inicio, fecha de final, fecha inicial permitida para apuntarse y fecha final permitida para apuntarse.

El usuario deberá rellenar los campos y pulsar el botón con el texto “crear actividad” que aparece en la parte inferior de la pantalla. El botón mientras se encuentre realizando la transacción en Ethereum mostrará el mensaje “procesando”. Una vez acabado mostrará el mensaje para informar al usuario que la transacción ha sido realizada satisfactoriamente.

Resulta importante rellenar todos los campos antes de enviar la información, si no el contrato inteligente no los aceptará y devolverá error. Las fechas deben cumplir que sean en un momento del futuro y que no se solapen entre ellas, es decir, la fecha final no puede ser anterior a la fecha inicial y el periodo para inscribirse a la actividad debe ser anterior al periodo en el que se realiza la actividad.

Para crear la actividad se muestra la interfaz de usuario mostrada en la figura 3.15.

5. Conclusiones

Durante el desarrollo del presente trabajo se han tenido que aprender gran cantidad de tecnologías. Estos conocimientos abarcan blockchain y Ethereum, creación de DApps y toda la tecnología que hay detrás. Esto, debido a las limitaciones de tiempo, se ha conseguido de forma básica. No obstante a pesar de las limitaciones se ha conseguido obtener un nivel en el que se cumplen sobradamente los objetivos iniciales.

Se debe tener en cuenta que además de la tecnología aprendida de forma teórica también se ha aprendido la forma práctica de poderlo aprovechar, construyendo y desarrollando una aplicación descentralizada basada en Ethereum que además se apoya en tecnologías punteras en el ámbito del desarrollo web como puede ser el manejo de React.js o Next.js. Asimismo Se han alcanzado aptitudes de desarrollo de contratos inteligentes mediante Solidity y cómo gestionar y desplegar un Smart Contract utilizando el entorno de ejecución Node.js.

Los objetivos se han cumplido, ya que se ha conseguido desarrollar una aplicación funcional y atractiva que utiliza como backend los Smart Contracts y Ethereum. Además se considera que el código entregado es de calidad, evitando anti patrones de diseño software así como eficiencia a la hora de su ejecución y desarrollo. Asimismo los objetivos de aprendizaje teórico se han podido obtener tal y como se puede ver en el capítulo del estado del arte, de tal forma que se han comprendido y aplicado a la hora del desarrollo de la aplicación. Finalmente primar la seguridad se ha tenido en cuenta desde el primer día de desarrollo del trabajo.

La planificación se ha seguido correctamente, si bien es cierto que al comienzo el diagrama de Gantt sufrió cambios, debidos fundamentalmente a la reestructuración de la planificación para poder atender de forma eficiente la redacción de la memoria y el desarrollo de la aplicación en sí, es decir, se decidió que el tiempo asignado al desarrollo de la aplicación por un lado y por otro lado del tiempo asignado a la redacción de la memoria se iban a llevar a cabo de forma concurrente. Los plazos se han cumplido por lo que la planificación ha cumplido de forma satisfactoria su objetivo.

Tal y como se ha dicho el trabajo tiene un tiempo limitado y siempre quedarán futuras mejoras pendientes para la aplicación. En este caso se han quedado en el tintero ideas como mejoras visuales de la aplicación, que, aunque se ha quedado satisfecho con el resultado y con la idea de cajas y una navegación intuitiva, siempre se puede mejorar y realizar animaciones o crear nuevas estructuras o componentes. Otra mejora sería la de la expansión de funcionalidad del usuario, esto es que no sea necesario que el creador de la aplicación deba añadir un nuevo usuario en el sistema, sino que se cree un sistema en el que automáticamente el usuario se cree su propia cuenta, con el envío de un correo y gestión de su cuenta de usuario. Finalmente indicar que se puede mejorar la forma en la que se muestran las actividades y añadir un buscador o paginación.

Las opciones de mejora son innumerables por lo que se han expresado las que se han considerado que revisten de mayor importancia de cara a añadir funcionalidad y que tengan un mayor recorrido. Por lo tanto se considera que se ha realizado un buen trabajo y que las opciones a futuro, tanto con la tecnología como con la aplicación desarrollada, poseen gran cantidad de diferentes vías y posibilidades abiertas.

6. Glosario

1. **Smart Contract o contrato inteligente.** Se trata de un programa y conjunto de normas que permite cumplir con una serie de acuerdos, y verificar que así sea, de forma automática. Sirve para definir una serie de normas y comportamientos que se puedan suscribir y que se pueda estar seguro que en caso de cumplir una serie de condiciones se van a llevar a cabo los acuerdos alcanzados predefinidos.

2. **DApp.** Aplicación descentralizada cuyo funcionamiento no necesita de terceras partes para funcionar y en los que la aplicación se define mediante la interacción de los diferentes usuarios que la utilizan. La aplicación se apoya en sistemas distribuidos y no ofrece una estructura central.

3. **Blockchain.** Tecnología que permite almacenar información utilizando cifrado y redes distribuidas que guardan cada una copias de información y que ayudan a crear una red resiliente que asegura, entre otras características, la integridad de los datos almacenados.

4. **Ethereum.** Tecnología descentralizada basada en blockchain que permite el desarrollo de aplicaciones distribuidas basadas en Smart Contracts.

5. **Compilador.** Programa que se ejecuta para convertir código fuente de una aplicación a código entendible por la máquina que lo debe ejecutar, ya sea código binario o estructuras intermedias como bytecodes.

6. **Frontend.** Parte visual de cualquier aplicación web. En ella se define la interfaz de usuario y el comportamiento de los elementos con respecto al usuario, así como las diferentes llamadas al backend de la aplicación.

7. **Backend.** Núcleo fundamental de cualquier aplicación web desde donde se define la funcionalidad de la aplicación.

8. **Gestor de paquetes.** Herramienta utilizada para la instalación y gestión de software y componentes externos que se desea añadir o gestionar en una determinada aplicación.

9. **ERC20.** Estándar para la construcción de tokens que permiten el intercambio entre diferentes tipos de forma sencilla, además de proveer de una interfaz para el manejo de los tokens.

10. **Token.** Representación de valor de un determinado activo en una aplicación.

7. Bibliografía

- [1] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. Bitcoin. https://bitcoin.org/bitcoin.pdf (acceso: 23 de mayo de 2021).
- [2]Roshan Raj. “What exactly is Blockchain mining?”. Intellipaat. https://intellipaat.com/blog/tutorial/blockchain-tutorial/what-is-bitcoin-mining/ (acceso: 23 de mayo de 2021).
- [3] F. Baothman, K. Saeedi, K. Aljuhani, S. Alkatheri, M. Almeatani et al., "Computational intelligence approach for municipal council elections using blockchain". *Intelligent Automation & Soft Computing*, vol. 27, no.3, pp. 625–639, 2021. [En línea]. Disponible en: https://www.techscience.com/iasc/v27n3/41658 . Acceso: 23 de mayo de 2021.
- [4] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends”, presentada en 6th IEEE International Congress on Big Data, Estados Unidos de América, pp. 557–564, 2017. [En línea]. Disponible en: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends
- [5] Vitalik Buterin. “Ethereum: The Ultimate Smart Contract and Decentralized Application Platform”. Internet Archive. http://web.archive.org/web/20131228111141/http:/vbuterin.com/ethereum.html (acceso: 23 de mayo de 2021).
- [6] Vitalik Buterin. “Ethereum Whitepaper”. Ethereum. https://ethereum.org/en/whitepaper/ (acceso: 23 de mayo de 2021).
- [7] Alex Preukschat. “Ethereum es Turing completo ¿y eso qué es?”. https://www.eleconomista.es/economia/noticias/8817210/12/17/Ethereum-es-Turing-completo-y-eso-que-es.html (acceso: 23 de mayo de 2021).
- [8] Fátima Leal, Adriana E. Chis y Horacio González–Vélez, “Multi-service model for blockchain networks”, *Information Processing & Management*, vol. 58, n.º 3, Mayo 2021. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0306457321000340# . Acceso: 23 de mayo de 2021.

- [9] Dr. Gavin Wood. “ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER”. Ethereum. https://ethereum.github.io/yellowpaper/paper.pdf (acceso: 23 de mayo de 2021).
- [10] Thomas Wiesner. “Ethereum Accounts, Addresses, Private and Public Keys”. Vomtom. https://vomtom.at/ethereum-private-and-public-keys/ (acceso: 23 de mayo de 2021).
- [11] Ethereum Foundation. (2021). Ether. Ethereum Homestead Documentation. https://ethdocs.org/en/latest/ether.html
- [12] Ethereum Foundation. (2021). Solidity. https://docs.soliditylang.org/en/latest/
- [13] Grant Bartel. “What is a Dapp? A Guide to Ethereum Dapps”. freeCodeCamp. https://www.freecodecamp.org/news/what-is-a-dapp-a-guide-to-ethereum-dapps/ (acceso: 23 de mayo de 2021).
- [14] Thomas Hay y Cheryl Douglass. “Ethereum JavaScript Libraries: web3.js vs. ethers.js (Part I)”. Infura blog. https://blog.infura.io/ethereum-javascript-libraries-web3-js-vs-ethers-js-part-i/ (acceso: 23 de mayo de 2021).
- [15] Gavin Wood. “Ethereum. How we arrived and where we’re heading”. Slideshare, diapositiva 42. https://i.stack.imgur.com/jzm8y.png (acceso: 23 de mayo de 2021).
- [16] Miguel A. Teruel y Juan Trujillo, “Easing ĐApp Interaction for Non-Blockchain Users from a Conceptual Modelling Approach”, *Applied Sciences*, vol 10, n.º 12, art. no. 4280, junio 2020. [En línea]. Disponible en: https://www.researchgate.net/publication/342385716_Easing_DApp_Interaction_for_Non-Blockchain_Users_from_a_Conceptual_Modelling_Approach . Acceso: 23 de mayo de 2021.

- [17] Jesús Correas, Pablo Gordillo y Guillermo Román-Díez, “Static Profiling and Optimization of Ethereum Smart Contracts Using Resource Analysis”, *IEEE Access*, vol. 9, pp. 25495-25507, 2021. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/9348894 . Acceso: 23 de mayo de 2021.
- [18] Fatmah Baothman, Kawther Saeedi, Khulood Aljuhani, Safaa Alkatheri, Mashael Almeatani y Nourah Alothman, “Computational Intelligence Approach for Municipal Council Elections Using Blockchain”, *Intelligent Automation & Soft Computing*, vol. 27, n.º 3, pp. 625–639, 2021. [En línea]. Disponible en: https://www.techscience.com/iasc/v27n3/41658 . Acceso: 23 de mayo de 2021.
- [19] Giuseppe Antonio Pierro, Roberto Tonelli y Michele Marchesi, “An Organized Repository of Ethereum Smart Contracts Source Codes and Metrics”, *Future Internet*, vol. 12, n.º 11, pp. 197-210, 2020. [En línea]. Disponible en: https://www.mdpi.com/1999-5903/12/11/197#cite . Acceso: 23 de mayo de 2021.
- [20] W3Schools. “JavaScript History”. W3Schools. https://www.w3schools.com/Js/js_history.asp (acceso: 23 de mayo de 2021).
- [21] Facebook Inc. “React”. React. https://reactjs.org/ (acceso: 23 de mayo de 2021).
- [22] OpenJS Foundation. “Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.”. Nodejs. https://nodejs.org/en/ (acceso: 23 de mayo de 2021).
- [23] Infura Inc. “Ethereum & IPFS APIs. Develop now on Web 3.0”. Infura. https://infura.io/ (acceso: 23 de mayo de 2021).
- [24] Nathan Reiff. “What Is ERC-20 and What Does It Mean for Ethereum?”. Investopedia. https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/ (acceso: 23 de mayo de 2021).
- [25] Bitcoinforme SL. “What is an ERC-20 token?”. Bit2Me Academy. https://academy.bit2me.com/en/what-is-erc-20-token/ (acceso: 23 de mayo de 2021).

- [26] OpenZeppelin. “openzeppelin-contracts”. Github. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol (acceso: 23 de mayo de 2021).

8. Anexos

Repositorio GitHub

Frontend:
https://github.com/DavidGomezSancho/smartactivities/tree/main/frontend

Backend:
https://github.com/DavidGomezSancho/smartactivities/tree/main/backend

Contratos:
https://github.com/DavidGomezSancho/smartactivities/tree/main/backend/contracts

Dirección Smart Contract desplegado en Rinkeby

SmartActivities
0x21CE94e9A37D4543991ad7Fc52BA5c3509f06EEc