Universitat Oberta de Catalunya

uoc.edu

Plataforma de tokenización de activos “Tokenizr”

José Manuel Valhondo Monroy
Máster en Ciberseguridad y Privacidad.
Trabajo Final de Máster. Sistemas de Blockchain

Nombre Consultor/a: Alberto Ballesteros Rodríguez
Nombre Profesor/a responsable de la asignatura:Victor García Font

Fecha Entrega: Enero de 2022

B) GNU Free Documentation License (GNU FDL)

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Plataforma de tokenización de activos*
Nombre del autor:	*José Manuel Valhondo Monroy*
Nombre del consultor/a:	Alberto Ballesteros Rodríguez
Nombre del PRA:	*Victor García Font*
Fecha de entrega (mm/aaaa):	01/2022
Titulación:	*Máster en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Sistemas de Blockchain*
Idioma del trabajo:	***Español***
Palabras clave	*Tokenización, activos, plataforma*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El auge de las tecnologías blockchain, y el uso de las mismas para el desarrollo de aplicaciones descentralizadas, hace que el uso de esta tecnología vaya mas allá de únicamente la creación de criptomonedas, pudiendo utilizarse, entre otras muchas cosas, para reflejar activos del mundo real, digitalizándolos y convirtiéndolos en activos digitales divisibles y transferibles.

El objetivo del presente trabajo es el desarrollo de una plataforma en forma de dApp, que permita la creación de tokens , que representarán activos del mundo real, mediante el despliegue de Smart Contracts que sigan el estándar ERC721, y operen en la red Ethereum, tanto Mainnet como Testnets.

Además de la propia funcionalidad del despliegue parametrizado en la red Ethereum, se añaden otras funcionalidades que son de utilidad para el producto final, como son una interfaz para el protocolo IPFS para incorporar ficheros en la red de nodos descentralizados cuyas direcciones únicad (CID) puedan ser incorporadas en cada contrato desplegado, y un pequeño explorador de bloques implementado mediante las api de Moralis, que permitirá visualizar los Smart Contracts desplegados. Uniendo las funcionalidades de los diferentes casos de uso se obtiene una herramienta integral para el despliegue de forma parametrizada de Smart Contracts basados en el estándar ERC721.

La conclusión final ha sido satisfactoria en relación a disponer, en una misma plataforma, de herramientas para desplegar y visualizar Smart Contracts en la red Ethereum, así como para interactuar con el protocolo IPFS.

Abstract (in English, 250 words or less):

The rise of blockchain technologies, and their use for the development of decentralized applications, makes the use of this technology go beyond just the creation of cryptocurrencies, and can be used, among many other things, to reflect assets of the world real, digitizing them and converting them into divisible and transferable digital assets.

The objective of this work is the development of a platform in the form of Dapp, which allows the creation of tokens, which will represent assets in the real world, through the deployment of Smart Contracts that follow the ERC721 standard, and operate on the Ethereum network, both mainnet as testnets.

In addition to the functionality of the parameterized deployment in the Ethereum network, other functionalities are added that are useful for the final product, such as an interface for the IPFS protocol to incorporate files in the network of decentralized nodes whose unique addresses (CID) can be incorporated into each deployed contract, and a small block explorer implemented through the Moralis api, which will allow viewing the deployed Smart Contracts. Uniting the functionalities of the different use cases, a comprehensive tool is obtained for the parametrized deployment of Smart Contracts based on the ERC721 standard.

The final conclusion has been satisfactory in relation to having, in the same platform, tools to deploy and visualize Smart Contracts on the Ethereum network, as well as to interact with the IPFS protocol.

Índice

Lista de figuras

Tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

La tokenización de activos es la forma en que cualquier activo del mundo real, tangible o intangible, se digitaliza y luego se divide en partes más pequeñas que toman la forma de tokens. En un sentido amplio, un token es básicamente la representación de otra cosa, y en este caso, cada token representa una parte proporcional del activo digitalizado.

Esto también significa que el propietario del token posee los derechos de propiedad u otro tipo de derechos del activo o de una parte del activo. La creación de este tipo específico de tokens, que se ejecutan en Smart Contracts.

Se pretende crear una plataforma que permita la tokenización de activos del mundo real, mediante la implementación de tokens NFT, basados en el estándar ERC721, generando para ello contratos inteligentes que persistirán en la red Blockchain de Ethereum.

Actualmente, sin tener en cuenta la utilización de esta tecnología, se puede recurrir a dividir un activo en participaciones o títulos, o incluso acciones, dependiendo siempre de un ente mas o menos centralizado, donde se negocian de forma pública esas acciones, aunque también pueden negociarse de forma privada. En definitiva un proceso que puede resultar lento, con inseguridades o con excesiva burocracia. Además de ser un proceso en el cual intervienen muchos actores e intermediarios lo que puede llegar a complicar el proceso.

Entre los beneficios de la tokenización de activos se encuentran principalmente los siguientes:

- La tokenización de activos puede crear nuevos modelos comerciales y sociales, como compartir la propiedad de la propiedad en sí o de los derechos que le pertenecen.

- La tokenización de activos solo puede existir utilizando la tecnología blockchain y, por lo tanto, esto significa que se utiliza una infraestructura compartida entre todos los participantes. Además, el hecho de que esté descentralizado, lo que significa que no requiere un tercero central, los costos de transacción se reducen significativamente. Esta descentralización mejora la eficiencia a la hora de realizar acciones, ya que la falta de un intermediario hace que se puedan crear automatizaciones.

- Una de las características principales del uso de blockchain como tecnología subyacente es su transparencia por defecto, ya que cualquier

persona en cualquier momento puede ver las transacciones realizadas en dicha blockchain.

- La capacidad de realizar transacciones fácilmente identificables agrega la ventaja de tener un solo punto de entrada para obtener información. Es más factible obtener toda la información en una sola llamada, que tener que depender de diferentes puntos de datos para verificar las transacciones y la propiedad.

- Además, dado que los datos almacenados en la cadena de bloques son inmutables, la misma información siempre está disponible para todos y la misma información estará disponible en cualquier momento.

En definitiva, el resultado que se desea obtener es una plataforma, a través de la cual, se podrán 'tokenizar' activos del mundo real, que una vez digitalizados, se indexarán en una blockchain y permitirá transacciones de los tokens del activo, con un nivel alto de transparencia, seguridad y trazabilidad, de tal forma que la prueba de propiedad de esos activos es patente y visible por cualquiera que necesite dicha verificación de propiedad.

1.2 Objetivos del Trabajo

Los principales objetivos del trabajo de fin de máster desarrollado, los se pueden dividir en tres categorías y son las siguientes:

1.2.1.-Investigación lenguajes de desarrollo de Smart Contracts.

En esta fase del proyecto se plantea como objetivo final familiarizarse con la tecnología en la cual se va a desarrollar la plataforma, mas concretamente:

- Investigación y tutoriales con lenguaje Solidity para desarrollo de Smart Contracts en Ethereum, así como otros frameworks y librerías que se utilizarán en el desarrollo:
  - TruffleSuite
  - Ganache
  - Web3js
  - Metamask (firmas de transacciones)
  - ...

- Investigación y tutoriales para el despliegue programático de Smart Contracts en Testnet y Mainnet de Ethereum con librerías We3.

- Generación de prototipos o pilotos de cara a la fase de desarrollo, así como una maqueta sin funcionalidad del front-end.

1.2.2.-Desarrollo de la Plataforma

En esta fase, que se solapa en el tiempo con la fase anterior de investigación, se realizará el desarrollo de la plataforma para la tokenización de los activos. En esta fase a su vez se diferenciarán varios hitos intermedios, algunos de los cuales coincidirán con las entregas de las PEC de la asignatura:

- Desarrollo de back-end, para el cual se utilizarán librerías para compilación de contratos inteligentes así como web3 para interactuar con la cadena de bloques. Se utilizará el lenguaje para especificación de contratos Solidity.

- Desarrollo de front-end: Desarrollo de la parte Front de la plataforma, para la cual se utilizarán principalmente frameworks y librerías de código abierto y gratuitas, los recursos software están descritos en el punto 1.4 de este plan de trabajo.

- Pruebas de back-end, para las cuales se utilizarán librerías ya conocidas para el testeo de contratos inteligentes como por ejemplo Mocha JS o Chai JS.

- Pruebas de front-end, se realizarán las pruebas de la parte Front mediante las librerías existentes en el framework de desarrollo Front, en nuestro caso Angular.

1.2.3.-Entrega

En esta fase, que se puede ir solapando con la fase anterior de desarrollo de la plataforma, se desarrollarán una serie de productos destinados a la entrega final del TFM, y también en cada una de las entregas parciales o PEC de la asignatura. Dentro de los objetivos de entrega se pueden describir los siguientes:

- Desarrollos de entregas parciales (PEC) definidas como hitos en el calendario de tareas.

- Desarrollo incremental de la memoria final, cumplimentándose a medida que se realizan las entregas y se definen los sprints.

- Realización del video de presentación del TFM.

- Defensa del TFM

Con todo lo anterior, una vez conseguidos los objetivos de investigación, desarrollo y entregas se obtiene un producto final, que tiene la capacidad de tokenizar activos del mundo real, previamente digitalizados, mediante la creación de Smart Contracts que se desplegarán en la red blockchain de Ethereum.

1.3 Enfoque y método seguido

Para llegar a los objetivos propuestos se opta por el desarrollo de la plataforma basándose en “esqueletos” de dApps propios de la suite Truffle denominados TruffleBoxes, que contienen una serie de funcionalidades mínimas ya implementadas, como pueden ser la conexión con wallets, y a partir de estos esquemas base se desarrolla la plataforma.

En cuanto a la metodología elegida para el desarrollo se opta por un desarrollo “agile” y a utilizar metodologías retro-alimentadas continuamente por el cliente, orientadas a sprints o fases con desarrollo de items pre-acordados, en las cuales, aunque el ambiente de estas metodologías tiende a absorber los cambios con facilidad, existen una serie de tareas o fases en las cuales se entregan una serie de características totalmente funcionales, que son prefijados y que no se pueden modificar hasta terminar dicha fase.

A estas fases se les denomina Sprints, (en la metodología SCRUM) y suelen durar de 2 a 4 semanas. Al terminar los sprints las partes (desarrolladores-clientes), evalúan la entrega y pueden surgir refinamientos que atañen únicamente a la parte de producto entregada en ese sprint, nunca serían nuevas funcionalidades. Algunos ejemplos de estas metodologías son SCRUM, KANBAN, XP,…

Para este proyecto, que requiere de mucho trabajo de desarrollo software y documentación, voy a utilizar una metodología ágil, mas concretamente metodología SCRUM, en la cual se fijarán una serie de sprints, cuyas entregas incrementales se harán coincidir con las entregas de PEC. En la figura siguiente se especifica a grandes rasgos la filosofía de esta metodología ágil.

Figura 1: Esquema de SCRUM

En esta metodología se especifica un repositorio de tareas/requerimientos, en el cual se indexan todas la funcionalidades o productos necesarios del proyecto, de tal forma que la suma de todos estos requerimientos constituye el producto final. En este repositorio, denominado Product Backlog, se incluirán requerimientos tanto de software como de documentación.

Se plantea por lo tanto para este proyecto la creación desde cero de un producto nuevo, desarrollado bajo la metodología indicada (SCRUM), para el cual se van a definir una serie de tareas que podrán obtener como productos, piezas de software con una entidad o autonomía suficiente, como para poder ser probadas de forma independiente, o elementos de documentación, que se incorporarán a la memoria final.

La entrega de estas tareas en cada hito (sprint), podrá venir seguida de una fase de retro-alimentación que depurará la entrega y contribuirá al desarrollo incremental. Las tareas que se entregarán en cada sprint, una vez se definan no podrán modificarse (sprint backlog), hasta finalizar el sprint. Si podrán, sin embargo, añadirse en mitad de un sprint nuevas tareas al backlog general.

Se plantea esta metodología en contraposición a otras de corte mas clásico como la metodología en cascada u otras como la estructurada, debido a que es mas tolerante a los cambios en los requerimientos y especificaciones y debido la poca experiencia del alumno en el ámbito de los contratos inteligentes, entendiendo que se evolucionará a partir de la especificación inicial del proyecto según avance en el estudio de las diferentes tecnologías que van a entrar en juego en el desarrollo del TFM.

Dado que la estructura del TFM se basa en entregas parciales, parece lo mas adecuado cuadrar este tipo de organización en las entregas ajustándolas a sprints, en la metodología SCRUM, en los cuales se cierran una serie de entregables para cada uno de ellos.

1.4 Planificación del Trabajo

Se especifican a continuación una serie de tareas y recursos que se han utilizado para el desarrollo del TFM, diferenciados en una serie de categorías que son:

- Recursos hardware
- Recursos software
- Identificación de tareas/sprints
- Productos entregados en cada sprint
- Product backlog.

Se especifica a continuación, de forma pormenorizada cada una de las categorías de recursos, tareas/sprints, que se han llevado a cabo para el desarrollo del TFM:

1.4.1.Recursos hardware

Recursos hardware	Motivo	Coste
Ordenador portátil personal	Ordenador principal para desarrollo de la plataforma, así como redacción de la memoria.	700€
Conexión a internet	Conexión a internet para descarga de dependencias, etc.	41€/mes
Servidor de backup (Sinology)	Servidor en el cual se mantendrá el repositorio del código de la plataforma, así como el product backlog, y podrá gestionarse cada sprint-backlog, mediante una instalación de GitLab.	800€

Tabla 1: Recursos hardware

1.4.2.Recursos software

Recursos software	Motivo	Coste
Visual Studio Code	IDE de desarrollo para la plataforma de contratos inteligentes	0€
Librerías Truffle	Librerías y frameworks necesarios para el desarrollo tanto de front-end como de contratos inteligentes	0€
Librerías Web3		0€
Librerías Angular		0€
Librerías Material UI		0€
Ganache	Simulador de Blockchain de Ethereum	0€
GitLab	Repositorio de código, para mantener un backup y versionado de cada una de las entregas. Este repositorio tiene características que permitirán gestionar tareas o funcionalidades a realizar del product backlog gestionándose como tarjetas en paneles.	0€
API Moralis	Librerías para obtener información de las diferentes testnets o mainnets.	0€
API IPFS	Librerías para interactuar con la red de nodos IPFS	0€

Tabla 2: Recursos software

1.4.3.Identificación de tareas/sprints en fechas

Como se ha comentado anteriormente, se ha seguido una metodología basada entregas o sprints, que se han cuadrado en el tiempo con las entregas parciales ya predefinidas en la asignatura. Las tareas identificadas en sprints son las siguientes:

Identificación de tareas/sprints	Inicio	Final
Mini-sprint 0- PEC1 – Plan de trabajo	20/09/2021	28/09/2021
Sprint 1 – PEC2 – Entrega seguimiento	29/09/2021	26/10/2021
Sprint 2 – PEC3 – Entrega seguimiento	27/10/2021	23/11/2021
Sprint 3 – PEC4 – Memoria Final	24/11/2021	28/12/2021
Sprint 4 – Presentación Video - Defensa	29/12/2021	14/01/2022

Tabla 3: Tabla 3: Identificación de sprints

1.4.4.Productos entregados en cada sprint

Identificación de tareas/sprints	Productos
Mini-sprint 0 - PEC1 – Plan de trabajo	Plan de trabajo
Sprint 1 – PEC2 – Entrega seguimiento	Piloto de desarrollo Smart Contracts + Piloto despliegue programático
Sprint 2 – PEC3 – Entrega seguimiento	Entrega Back-end
Sprint 3 – PEC4 – Memoria Final	Memoria Final + Producto completo
Sprint 4 – Presentación Video - Defensa	Entrega video presentación + Defensa final

Tabla 4: Productos en cada sprint

1.4.5.Product backlog

A continuación se especifica la relación de elementos, tareas y requerimientos que configurarán el Product Backlog que, tal y como se ha explicado en la definición de la metodología a utilizar en el TFM, puede ir modificándose a lo largo del tiempo de desarrollo del proyecto.

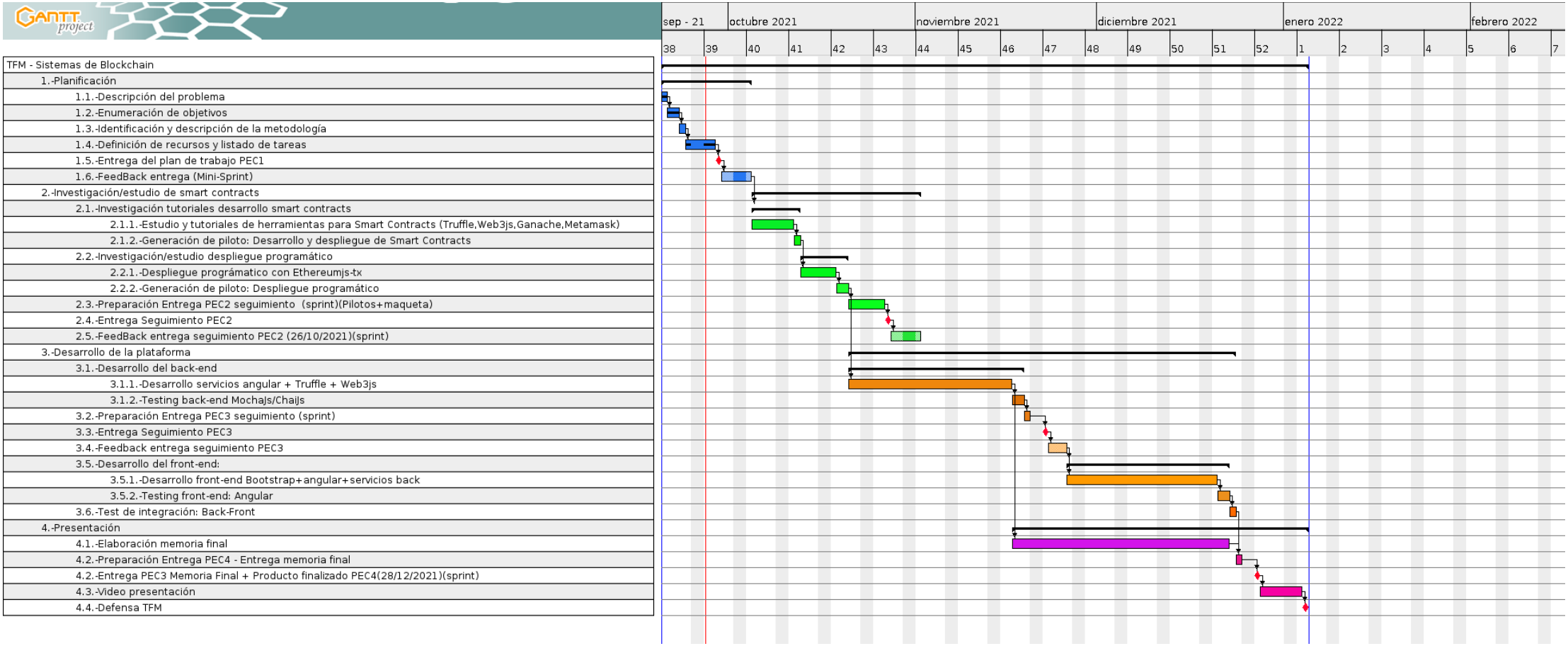
Tarea/Requerimiento	Tipo
Plan de trabajo (PEC1)	Documentación
Piloto Desarrollo y despliegue de Smart Contracts	Software

Piloto despliegue de contratos programática-mente	Software
Documentación (PEC2)	Documentación
Documentación (PEC3)	Documentación
Maqueta Front-end Html: Angular+Material UI	Software
Memoria Final (PEC4)	Documentación
Video presentación	Software
Desarrollo Plataforma: Front-end	Software

Tabla 5: Product Backlog

1.4.6.Diagrama de Gantt

A continuación se especifica una planificación basada en un diagrama de Gantt con la planificación:



1.5 Planificación

Nombre de la tarea	Fecha de inicio	Fecha de fin	Duración
TFM - Sistemas de Blockchain	20/09/21	5/01/22	77
1.-Planificación	20/09/21	5/10/21	11
1.1.-Descripción del problema	20/09/21	21/09/21	1
1.2.-Enumeración de objetivos	21/09/21	23/09/21	2
1.3.-Identificación y descripción de la metodología	23/09/21	24/09/21	1
1.4.-Definición de recursos y listado de tareas	24/09/21	29/09/21	3
1.5.-Entrega del plan de trabajo PEC1	29/09/21	30/09/21	1
1.6.-FeedBack entrega (Mini-Sprint)	30/09/21	5/10/21	3
2.-Investigación/estudio de smart contracts	5/10/21	2/11/21	20
2.1.-Investigación tutoriales desarrollo Smart Contracts	5/10/21	13/10/21	6
2.1.1.-Estudio y tutoriales de herramientas para Smart Contracts (Truffle,Web3js,Ganache,Metamask)	5/10/21	12/10/21	5
2.1.2.-Generación de piloto: Desarrollo y despliegue de Smart Contracts	12/10/21	13/10/21	1
2.2.-Investigación/estudio despliegue programático	13/10/21	21/10/21	6
2.2.1.-Despliegue programático con Web3	13/10/21	19/10/21	4
2.2.2.-Generación de piloto: Despliegue programático	19/10/21	21/10/21	2
2.3.-Preparación Entrega PEC2 seguimiento (sprint)(Pilotos+maqueta)	21/10/21	27/10/21	4
2.4.-Entrega Seguimiento PEC2	27/10/21	28/10/21	1
2.5.-FeedBack entrega seguimiento PEC2 (26/10/2021)(sprint)	28/10/21	2/11/21	3
3.-Desarrollo de la plataforma	21/10/21	24/12/21	46
3.1.-Desarrollo del back-end	21/10/21	19/11/21	21
3.1.1.-Desarrollo servicios angular + Truffle + Web3js	21/10/21	17/11/21	19
3.1.2.-Testing back-end MochaJs/ChaiJs	17/11/21	19/11/21	2
3.2.-Preparación Entrega PEC3 seguimiento (sprint)	19/11/21	20/11/21	1
3.3.-Entrega Seguimiento PEC3	22/11/21	23/11/21	1
3.4.-Feedback entrega seguimiento PEC3	23/11/21	26/11/21	3
3.5.-Desarrollo del front-end:	26/11/21	23/12/21	19
3.5.1.-Desarrollo front-end Bootstrap+angular+servicios back	26/11/21	21/12/21	17
3.5.2.-Testing front-end: Angular	21/12/21	23/12/21	2
3.6.-Test de integración: Back-Front	23/12/21	24/12/21	1
4.-Presentación	17/11/21	5/01/22	35
4.1.-Elaboración memoria final	17/11/21	23/12/21	26
4.2.-Preparación Entrega PEC4 - Entrega memoria final	24/12/21	25/12/21	1
4.2.-Entrega PEC3 Memoria Final + Producto finalizado PEC4(28/12/2021) (sprint)	27/12/21	28/12/21	1
4.3.-Video presentación	28/12/21	4/01/22	5
4.4.-Defensa TFM	4/01/22	5/01/22	1

Tabla 6: Tareas en la planificación

1.6.Análisis de riesgos

En este punto se identifican una serie de riesgos que pueden influir en el desarrollo del proyecto y que deberían ser tenidos en cuenta, se especifica para cada uno de ellos una pequeña descripción del riesgo identificado y las posibles medidas de mitigación que se deberán tener en cuenta para reducir al máximo el riesgo:

R1.- Falta de experiencia en la tecnología del desarrollador.

Descripción del riesgo: Falta de experiencia del desarrollador en la tecnología utilizada para el desarrollo de la plataforma, que pueda ralentizar la fase de desarrollo y causar un retraso al resto de fases.

Mitigación: Ampliar las fases de investigación y aprendizaje del lenguaje, herramientas y frameworks en caso necesario.

R2.- Problemas por incompatibilidades entre las tecnologías escogidas

Descripción del riesgo: Posibles problemas de incompatibilidades entre los lenguajes, herramientas o frameworks elegidos para el desarrollo del proyecto, por falta de experiencia en el desarrollo con este tipo de herramientas.

Mitigación: Realizar una fase de elección de las herramientas y librerías en la que se tenga un especial cuidado en incompatibilidades y tener herramientas o librerías de "reserva", para el caso de que algunas de las elegidas no fuesen compatibles con el resto de herramientas del proyecto.

R3.- Falta de tiempo para finalizar el desarrollo

Descripción del riesgo: Falta de tiempo para el desarrollo de la plataforma, según se ha planificado inicialmente, tanto para la parte back como la parte front-end.

Mitigación: Ajustar los tiempos de las fases de investigación y no demorar en exceso el inicio del desarrollo de la plataforma, o incluso adelantar el inicio del desarrollo a lo especificado en la planificación.

R4.- Falta de tiempo para redactar la memoria final

Descripción del riesgo: Falta de tiempo para desarrollo de la memoria final, debido a que otras fases se alarguen mas de lo planificado en un principio.

Mitigación: Documentar todo el proceso llevado a cabo durante el desarrollo de la plataforma, para que a la postre dicha documentación pueda formar parte de la memoria final.

Este análisis de riesgos se podrá ampliar y revisar para fases mas tardías del proyecto según se vayan alcanzando hitos.

1.7 Breve sumario de productos obtenidos

Los productos obtenidos al finalizar el desarrollo de este trabajo son los siguientes:

- **Aplicación descentralizada para tokenización de activos**: Aplicación implementando distintas funcionalidades, entre ellas una interfaz para tokenización de activos parametrizables, que desplegarán Smart Contracts en la red Ethereum (Alojada en un repositorio)
- **Repositorios de código ejemplos y POC**: A lo largo del desarrollo del proyecto, se han realizado una serie de pruebas de concepto y contratos de prueba. Estos repositorios se especifican en este documento.
- **Memoria Final**: Memoria final del TFM.

1.8 Breve descripción de los otros capítulos de la memoria

En los siguientes capítulos de la memoria, se hace una contextualización y estado del arte de la tokenización de activos mediante Smart Contracts NFT (ERC721), así como una definición a bajo nivel de la arquitectura de la solución implementada, ahondando en las distintas funcionalidades de los casos de uso, así como en las API tanto libres como comerciales que se utilizan para desarrollar dichas funcionalidades.

2.Tokenización de activos

2.1.Definición y contexto

Dependiendo del tipo de documentación que se consulte, la definición de token puede variar en su acepción. En 2017 William Mougayar realizó una definición en la cual identificaba un token como «*una unidad monetaria programable que está vinculada a una blockchain, y es parte de la lógica de los contratos inteligentes en el contexto de una aplicación de software específica*» [W-5].

De igual manera, en el mismo artículo el autor analiza y describe las características y funcionalidades que tienen los tokens, dividiendo en una serie de roles, propósitos y características de estos activos. En la siguiente imagen se especifica para cada uno de los roles identificados por el autor un propósito y una serie de características

Figura 2: Imagen adaptada del artículo original en [5]

Al evaluar una determinada organización basada en tokens, cuantas más características se puedan marcar en relación con la función del token, mejor será éste, además resulta esencial conocer la naturaleza jurídica de lo que se desea tokenizar, ya que será la naturaleza jurídica- no la denominación- del activo la que determinará cuál es su régimen de propiedad, de transmisibilidad o de responsabilidad [W-7].

Dependiendo del activo– en sentido jurídico- que se esté tokenizando, la relación entre el token y el poseedor- y entre el token y terceros- será distinta. Y lo cierto es que hay infinidad de categorías de cosas desde el punto de vista

jurídico o tecnológico. En todo caso, los activos - y, por tanto, los tokens- deben guardar siempre los siguientes atributos:

1. Utilidad: ha de tener un valor económico o, al menos, un valor susceptible de satisfacer alguna necesidad humana.

2. Sustantividad: debe tener una existencia autónoma.

3. Apropiabilidad (física o jurídica): ha de ser susceptible de sumisión jurídica a su titular.

4. Comerciabilidad: es discutido en la doctrina general; sin embargo, en el caso de la tokenización de cosas, parece obvia esta característica.

Una vez expuesto todo lo anterior, el proceso de tokenización no es mas que, emitir tokens en una blockchain, mediante los cuales se puede representar de forma digital cualquier tipo de activo.

2.1.1. Initial Coin Offering

En el mundo de las criptomonedas o criptodivisas se utiliza un instrumento que se conoce como ICO (Initial Coin Offering en inglés) para financiar el desarrollo de nuevos protocolos. Esta expresión, traducida literalmente al castellano significa Oferta Inicial de Moneda.

Teniendo en cuenta las siguientes características de los tokens :

- Los tokens se crean con las mismas características de seguridad y descentralización que las criptomonedas como el bitcóin; siguiendo estándares mínimos de interoperabilidad, lo que los hace fácilmente transferibles y negociables.

- Los tokens representan activos, físicos o virtuales, o incluso sirven como ficha de cambio para el uso de servicios en nuevos modelos de negocio empresarial.

Entonces, es posible intuir que los tokens deberían tener un valor, al menos, similar al del activo que representan. Teniendo todo esto en cuenta, en muchos casos los tokens están sirviendo incluso para la financiación inicial de algunas iniciativas empresariales de nueva creación. En las llamadas Initial Coin Offering (ICO) una empresa vende una cantidad limitada de tokens de nueva creación en un estado muy inicial del negocio. Así, mediante este sistema de financiación parecido a lo que sería una Initial Public Offering (IPO) en el mundo industrial, la empresa recauda fondos de los inversores en una modalidad parecida al crowdfunding sin necesidad de acudir a bancos o a venture capitalists, y ahorrándose multitud de procesos requeridos para la obtención tradicional de financiación [W-8].

Por otro lado, los inversores confían en que en cuanto el proyecto vinculado al token que han adquirido despegue, el token se revalorice y, si lo

desean, puedan aprovechar los mercados de intercambio y la liquidez de las criptomonedas para obtener beneficio de los tokens adquiridos [D-1].

2.1.2. Tipos de tokens

La principal categorización de tokens se puede realizar atendiendo a su funcionalidad, y se pueden dividir en: Utility tokens y Security Tokens.

Utility tokens: Los Utility tokens son, por lo general, tokens que permiten a su propietario acceder a servicios o usar productos ofrecidos por una empresa o por una aplicación descentralizada.

La creación de este tipo de token es muy propicia para, conseguir financiación para proyectos en estadios muy iniciales mediante las ICO. Así, sus promotores emiten utility tokens que dan derecho al uso futuro de una plataforma una vez desarrollada[D-1].

Security tokens: Partiendo del significado de la palabra «security »en el ámbito de las finanzas que viene a significar «valor», estos tokens representan un activo financiero intercambiable. Mediante tecnología blockchain se busca digitalizar activos de este tipo creando Security Tokens que permitan aprovechar las ventajas que ofrece esta tecnología reduciendo costes, eliminando intermediarios y burocracia[D-1].

2.2.Plataformas de tokenización

En una primera toma de contacto con las tecnologías que entran en juego para el desarrollo de este tipo de plataformas, se ha observado que existen multitud de tutoriales y ejemplos de desarrollo de Smart Contracts para la red Ethereum, sin embargo después de una búsqueda de plataformas de tokenización disponibles la mayoría de las que se han encontrado son plataformas de empresas que ofrecen este tipo de servicio previo pago y son plataformas propietarias como por ejemplo:

- LandToken (https://www.landtoken.net/)
- Brickken (https://www.brickken.com/es/)
- Bridge Protocol (https://bridge.mtpelerin.com/)
- Tokeniza (https://www.tokeniza.es/)

Examinando estas ofertas de tokenización se puede observar, que ninguna de ellas realmente ha despegado, tal vez porque ninguna de estas plataformas ofrece o puede ofrecer el ciclo completo, ya que, la tokenización del activo y la inclusión en la cadena de bloques, es una de las partes a las que se deben añadir una correcta estructuración legal, gestionar los mercados, custodia de tokens, liquidez,etc.

Esto afecta a que los posibles inversores en una oferta de tokens tienen que depender de varios proveedores si quieren ejecutar ordenes relativamente complejas.

Si se consiguiese unificar en una misma interfaz de usuario, la gestión de mercados de activos tokenizados, la creación y custodia de tokens, con la gestión propia de los mercados de intercambio tradicionales, es posible que se pudiese ver un incremento en la participación de inversores en este tipo de activos.

2.2.Smart Contracts ERC721

Un token no fungible (NFT) se usa para identificar algo o alguien de una manera única. Este tipo de Token es perfecto para ser utilizado en plataformas que ofrecen artículos coleccionables, claves de acceso, boletos de lotería, asientos numerados para conciertos y partidos deportivos, etc. Este tipo especial de Token tiene una serie de posibilidades y características que se vieron plasmadas en un estándar, mas concretamente el ERC-721[W-9].

El ERC-721 introduce un estándar para NFT, es decir, este tipo de Token es único y puede tener un valor diferente al de otro Token del mismo Smart Contract, quizás por su antigüedad, rareza o incluso algo más trivial como su apariencia visual [W-9].

El ERC-721 (Ethereum Request for Comments 721), propuesto por William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs en enero de 2018, es un estándar de tokens no fungibles que implementa una API para tokens dentro de Smart Contracts.

Proporciona funcionalidades como transferir tokens de una cuenta a otra, obtener el saldo actual del token de una cuenta, obtener el propietario de un token específico y también el suministro total del token disponible en la red. Además de estas, también tiene algunas otras funcionalidades como aprobar que una cantidad de token de una cuenta pueda ser movida por una cuenta de terceros.

2.3.Otros contratos

Aunque el trabajo desarrollado se centra en el despliegue de Smart Contracts siguiendo el estándar ERC-721 para NFT, durante la fase de estudio de Smart Contracts se han observado otra serie de estándares de contratos que amplían o tomar elementos de ERC-721 y de otros estándares y pueden tener interés a la hora de su implementación, estos son:

2.3.1.-ERC-1155 (Multi-token)

La característica distintiva de ERC1155 es que utiliza un solo contrato inteligente para representar múltiples tokens a la vez. Esto es similar a cómo ERC721 hace las cosas, pero en ese estándar, un ID de token no tiene un concepto de equilibrio: cada token no es fungible y existe o no. Por otro lado, en ERC1155 las cuentas tienen un saldo distinto para cada ID de token, y los tokens no fungibles se implementan simplemente acuñando uno de ellos.

Este enfoque conduce a ahorros masivos de gas para proyectos que requieren múltiples tokens. En lugar de implementar un nuevo contrato para

cada tipo de token, un solo contrato de token ERC1155 puede mantener todo el estado del sistema, lo que reduce los costos de implementación y la complejidad [W-18].

2.3.2.-Contrato Ricardiano

Un Contrato Ricardiano no tiene porqué ser en si un Smart Contract, por definición es una forma de documentos digitales que actúan como un acuerdo entre dos partes sobre los términos y condiciones para una interacción entre las partes acordadas.

Lo que lo hace único es que está firmado y verificado criptográficamente. Incluso cuando se trata de un documento digital, está disponible en un texto legible para personas que también es fácil de entender para aquellos que no son abogados. Es un acuerdo o documento legal único que se puede leer al mismo tiempo para los programas informáticos y para los humanos.

En pocas palabras, tiene dos partes o dos propósitos. Primero, es un contrato legal fácil de leer entre las dos o más partes. Tu abogado puede entenderlo fácilmente, e incluso tú puedes leerlo y entender los términos básicos del contrato.

Segundo, también es un contrato legible para máquinas. Con las plataformas de blockchain, estos contratos ahora se pueden identificar mediante hashes o funciones resumen, firmar y guardar fácilmente en la blockchain.

En definitiva, los contratos Ricardianos combinan contratos legales con tecnología, y la tecnología de la blockchain mas concretamente [W-19].

3.Estado del arte

En este punto se especifican las herramientas disponibles para el desarrollo del proyecto a nivel de IDEs de desarrollo, librerías y herramientas externas que componen el ecosistema para desarrollo de aplicaciones descentralizadas basadas en Smart Contracts, como es el caso de la plataforma desarrollada en este TFM. Se especificara también cuales de ellas han sido definitivamente utilizadas en el desarrollo justificando su utilización.

3.1.- IDE de desarrollo.

Para el desarrollo de la plataforma de tokenización existen dos ámbitos de programación claramente diferenciados, por un lado el desarrollo del contrato o contratos que posteriormente se desplegarán en las redes tanto de prueba como Mainnets, y por otra parte el desarrollo de la dApp que gestionará los parámetros y atributos que se van a pasar al contrato desarrollado, parametrizando éste, previo al despliegue.

3.1.1.-Remix

Para comenzar con el desarrollo de Smart Contracts una herramienta que favorece bastante el desarrollo, despliegue e interacción con los Smart Contracts es Remix, ya que es posible desarrollar contratos y desplegarlos de una forma sencilla para posteriormente interactuar con ellos, y permite hacerlo tanto en redes principales (vía web3) como en redes de prueba en modo Sandbox. En esta fase del proyecto se ha utilizado este IDE para varios de los ejemplos y tutoriales que se han realizado. Para la utilización de los Smart Contracts desarrollados en la herramienta en el desarrollo de dApps, sería necesario

3.1.2.-Visual Studio Code

Esta herramienta es un IDE multipropósito que puede ser ampliado en base a plugins, los cuales le permiten añadir muchas funcionalidades y compatiblidad con el desarrollo en casi cualquier lenguaje. Para el caso del desarrollo de Smart Contracts en Solidity existen plugins que añaden funcionalidad de compilación dentro del propio IDE.

3.2.- Librerías y Frameworks

Para el desarrollo de aplicaciones descentralizadas se hacen necesarias una serie de acciones, funcionalidades que se repiten en todas ellas o en la mayoría (conexión con la wallet, instancia de un contrato, ejecución de los métodos del contrato, etc), en lugar de implementar cada proyecto cada una de estas funcionalidades de forma independiente se han estandarizado, “de facto”, una serie de librerías y frameworks que unifican el proceso de realizar estas tareas comunes.

En este punto se va a mostrar una relación de las librerías, frameworks y herramientas que permiten el desarrollo “acelerado” de dApps debido a la estandarización de la realización de las tareas mas comunes.

3.2.1.Truffle

Truffle es un entorno de desarrollo, un marco de prueba y una canalización de activos para cadenas de bloques que utilizan la máquina virtual Ethereum (EVM), con el objetivo de facilitar la vida como desarrollador[W-14].

Con Truffle se pueden realizar las siguientes tareas:

- Compilación, vinculación, implementación y gestión binaria de contratos inteligentes integrados.
- Pruebas de contrato automatizadas para un desarrollo rápido. Marco de migraciones y despliegue extensible y programable.
- Gestión de red para implementar en cualquier número de redes públicas y privadas.
- Gestión de paquetes con EthPM & NPM, utilizando el estándar ERC190.

- Consola interactiva para comunicación directa por contrato.
- Canal de compilación configurable con soporte para una integración estrecha.
- Ejecutor de scripts externo que ejecuta scripts dentro de un entorno Truffle.

3.2.2.TruffleBox

En los TruffleBox, perteneciente a la suite de Truffle, se mezclan proyectos Truffle (para desarrollo de Smart Contracts) con proyectos de tecnologías web de todo tipo, principalmente basadas en JavaScript (Angular, ReactJS,...).

Son arquetipos de proyectos creados por la comunidad, en los cuales se ofrece una estructura base para crear dApps basadas en Truffle y se le añaden estructuras de aplicaciones web para la creación de Front-ends de dichas dApps, así como librerías y herramientas útiles para el desarrollo de este tipo de aplicaciones (WalletConnect, AngularForms,..).

3.2.3.OpenZeppelin

OpenZeppelin proporciona un conjunto completo de productos de seguridad para crear, administrar e inspeccionar todos los aspectos del desarrollo y las operaciones de software para proyectos de Ethereum.

Ofrece un conjunto de librerías para el desarrollo seguro de Smart Contracts que ofrece ademas una serie de Smart Contracts reutilizables que pueden ser utilizados para implementar contratos para muchos estándares de Ethereum (ERC721, ERC20,ERC777,...)

3.2.4.Metamask

Esta aplicación permite interactuar con aplicaciones Blockchain, y asociar cuentas a dichas aplicaciones, ya que también tiene funcionalidades de Wallet (cartera) de criptomonedas. Con Metamask se puede conectar a multitud de redes Blockchain, tanto Mainnets como Testnets. Es una herramienta, por lo tanto, muy útil para los desarrolladores a la hora de probar sus dApps con redes blockchain en un entorno muy fiel al entorno de funcionamiento real.

Esta aplicación es ofrecida en formato aplicación nativa para móviles, y también como plug-in instalado en los distintos navegadores.

3.3.- Blockchain locales y testnets

En cualquier proyecto que requiera un desarrollo software es necesario, o al menos altamente recomendable, disponer de varios entornos de desarrollo, en los cuales se evoluciona la aplicación antes de someterla a un entorno real de producción. Se pueden configurar entornos de desarrollo, de pre-producción, de Integración, etc. Una de las características que deben tener todos estos entornos, es que deben tener cierta similitud con el entorno en el cual se desplegará la aplicación finalmente, esto depura errores que pueden

venir asociados al sistema operativo, máquina virtual, librerías etc, otorgando mas tiempo al desarrollador para pulir los errores que sean propios de la aplicación y pudiendo abstraerse este del entorno (Sistema operativo, versión de compilador, etc) en el que se esté desarrollando.

En el mundo blockchain también existen opciones para configurar estos entornos de prueba, que pueden estar online o bien que pueden ser desplegadas de forma local. A continuación se ofrecen una serie de Blockchains (Ethereum) de prueba tanto online como de forma local, que han sido tenidas en cuenta a la hora de realizar la implementación del proyecto.

3.3.1.- Redes Ethereum de pruebas

Además de la red principal, existen las redes de prueba públicas. Estas redes las utilizan los desarrolladores de protocolos o los desarrolladores de contratos inteligentes para probar las actualizaciones de los protocolos y los posibles Smart Contracts en un entorno similar a los entornos de producción antes de implementarlos en la red principal. Como ejemplo, en los servidores de producción frente a los de pre-producción.

La mayoría de las redes de prueba utilizan un mecanismo de consenso de prueba de autoridad. Es decir, se escoge un pequeño número de nodos para validar las transacciones y crear nuevos bloques apostando sus identidades en el proceso. Es difícil incentivar el minado en una red de pruebas con una Prueba de trabajo, ya que podría conllevar vulnerabilidades.

Algunas de estas redes de prueba son las siguientes:

- **Görli**: Red de pruebas con prueba de autoridad.
- **Kovan**: Red de pruebas para clientes de OpenEthereum.
- **Rinkeby**:Red de pruebas para clientes de Geth.
- **Ropsten**: Red de pruebas con prueba de trabajo, es la mas parecida a Ethereum debido a que usan el mismo método de consenso.

3.3.2.- Ganache

En el apartado anterior se especificaban una serie de redes Blockchain de prueba, que son muy útiles para el tiempo de desarrollo de los diferentes Smart Contracts y dApps, son diferentes entornos de Ethereum a los que se puede acceder para desarrollar, probar o producir casos de uso. El caso es que todas estas redes son redes públicas a las que todo el mundo puede acceder, desplegar sus contratos y realizar transacciones, y es necesario conseguir ETH de prueba para realizar todas estas transacciones. Para evitar el hecho de tener que realizar pruebas en redes públicas, que pueden estar mas o menos saturadas, y sobre todo de tener que conseguir ETH de prueba para realizar las transacciones, se puede desplegar una red blockchain de pruebas en local con Ganache[W-12], con la cual se dispondrá de toda la funcionalidad de estas redes pero en una red propia.

Ganache es una cadena de bloques personal para el desarrollo rápido de aplicaciones distribuidas de Ethereum y Corda. Se puede utilizar Ganache durante todo el ciclo de desarrollo; lo que le permite desarrollar, implementar y

probar las dApps en un entorno seguro y controlado. La aplicación se presenta en modo gráfico, como se puede observar en la imagen anterior, y también existe un modo consola.

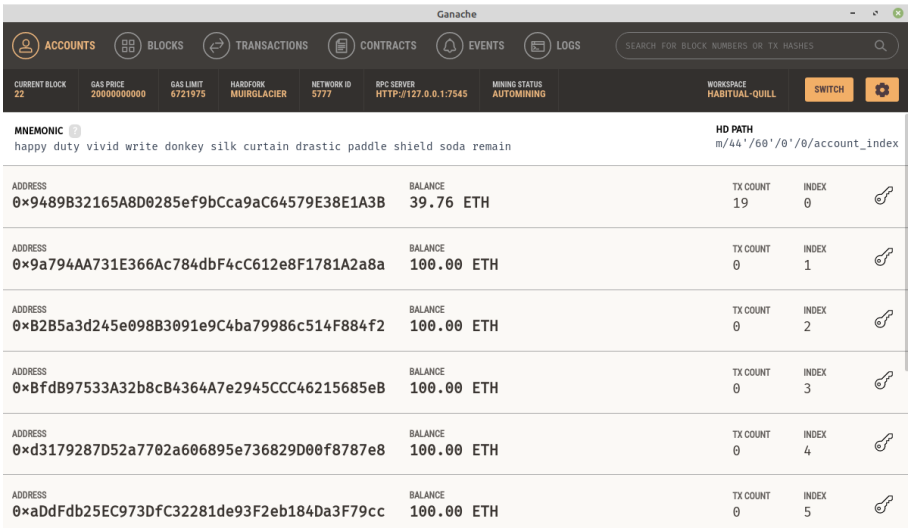


Figura 3: Ganache en modo escritorio

3.4.- IPFS

En el despliegue de los contratos a menudo va a ser necesario asociar ficheros o documentos a esos contratos, como por ejemplo en los NFT coleccionables, cada token está asociado a un archivo de imagen, video o audio. Para no incorporar en los propios contratos dichos items, ya que podría tener un coste muy alto, se hace uso de IPFS (Interplanetary File System) que es un protocolo de archivos distribuidos peer-to-peer , donde los archivos se almacenan asociados a un hash único que los identifica de manera global.

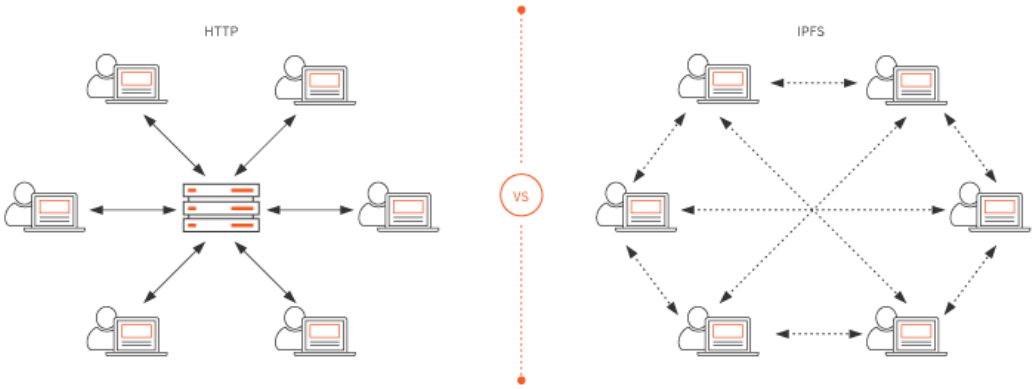


Figura 4: Diferencia entre protocolo http e ipfs. Fuente [W-10]

Este sistema asigna una dirección única a cada fichero o documento que se incorpora al protocolo, de tal forma que si se necesitase incorporar un documento a Smart Contract es mucho mas ágil incorporar estas direcciones únicas en el contrato y posteriormente interactuar con el contrato para obtener dicha dirección y descargar el fichero o documento.

Una forma de conectarse con un nodo IPFS es descargar IPFS en el sistema y ejecutarlo en la máquina para que sea un nodo IPFS, conectando posteriormente la dApp mediante métodos que ofrece el propio protocolo.

Otro método es utilizar una API pública como la que se ofrece en Infura.io [W-11], y mediante el uso de librerías conectarse para utilizar sus servicios.

3.5.-Exploración de bloques

Aunque el principal objetivo del proyecto es la construcción de una plataforma para tokenizar activos, existen otra serie de funcionalidades que le aportan valor al proyecto y que pueden ser de ayuda para poder visualizar, de forma adecuada, las modificaciones que se realizan sobre la Blockchain. Una de estas funcionalidades es la exploración de bloques, que es básicamente, acceder a la información que se almacena en la propia cadena de bloques para extraer de ésta información de transacciones, tokens asociados a wallets concretas, etc. Se presentan a continuación algunas alternativas para obtener información de la Blockchain, que tienen una amplia variedad de APIs y son utilizadas por miles de desarrolladores a nivel mundial, pero tienen el inconveniente de que son servicios centralizados, con lo que ello conlleva a nivel de ataques, escalabilidad, etc.

3.5.1.- Infura

La API de Infura Standard Ethereum proporciona acceso instantáneo a la red Ethereum a través de HTTPS y WebSocket. Proporciona una conexión accesible y confiable para los desarrolladores, lo que le permite dedicar menos tiempo a la infraestructura y más tiempo a crear software[W-13].

Tiene un plan basado en niveles, entre los que se encuentra un plan gratuito con unas 100000 peticiones al día y tres proyectos, con acceso a Mainnet y Testnets de Ethereum. Para conectar con esta plataforma se pueden utilizar librerías estándar que ya existen en los repositorios.

3.5.2.- Moralis

Según su la definición en su propio sitio web, Moralis proporciona backend administrado para proyectos de blockchain, además de sincronizar automáticamente los saldos de sus usuarios en la base de datos, lo que le permite configurar alertas en cadena, ver eventos de contratos inteligentes, crear índices y mucho más. Se accede a todas las funciones a través de un

SDK fácil de usar. Todas las funciones que ofrece Moralis son de cadena cruzada de forma predeterminada.

También tiene diferentes niveles de acceso en base a peticiones realizadas, tiempos de actividad del servidor, etc.

La gran diferencia observada con respecto al proveedor anterior, radica en que en este caso existen en los repositorios (npm) librerías propias que permiten un acceso mas sencillo a las funcionalidades de la API.

4.Arquitectura de la solución

Con todo lo expuesto anteriormente se dispone de toda la información para diseñar la plataforma. En este punto se expondrá una descripción de la misma, además de la estructura escogida para su implementación, así como su análisis y desarrollo basado en casos de uso con cierta independencia entre si. Por último se especificarán una serie de herramientas y API, escogidas para formar un stack de tecnologías de entre todas las expuestas en el punto anterior.

4.1.Descripción de la plataforma

Se pretende desarrollar una plataforma, mediante la cual se puedan generar tokens NFT en las redes de Ethereum (Mainnet / Testnets), mediante el despliegue de contratos inteligentes que sigan el estándar ERC721, y poder visualizar los tokens NFT desplegados en la red (Kovan), así como las transacciones realizadas por la cuenta conectada. También será necesario disponer de una interfaz para incorporar ficheros y documentos a la red IPFS cuyos identificadores únicos (CID) puedan incorporarse a los contratos desplegados.

La información de desarrollo de la plataforma son los siguientes:

Sistema Operativo: **Linux Mint 19.3. Cinnamon**
Procesador: **Intel i7 8th Gen**,
RAM: **24Gb RAM**.
Versión NPM: **6.14.15**
Version NodeJs: **14.18.1**
Versión Material UI:
IDE: **Visual Studio Code 1.53**.
Chrome: **87.0.4280.141**
Versiones Angular:

Angular CLI: 12.2.10

@angular-devkit/architect	**0.1202.10 (cli-only)**
@angular-devkit/core	**12.2.10 (cli-only)**
@angular-devkit/schematics	**12.2.10 (cli-only)**
@schematics/angular	**12.2.10 (cli-only)**
@angular/material	**11.2.1**

4.2.Estructura del sistema

El sistema se presenta como una aplicación descentralizada, desarrollada en Angular , basada en un esqueleto TruffleBox [W-15], del cual se han mantenido las funcionalidades de conexión con wallet a través de Metamask o WalletConnect. Los diferentes casos de uso se harán coincidir con componentes Angular independientes a nivel funcionalidad, que a su vez utilizarán servicios de conexión con las diferentes API o servicios que se utilizan (IPFS-Infura y Moralis). El esquema de conexiones de la aplicación es el que se muestra en la imagen siguiente:

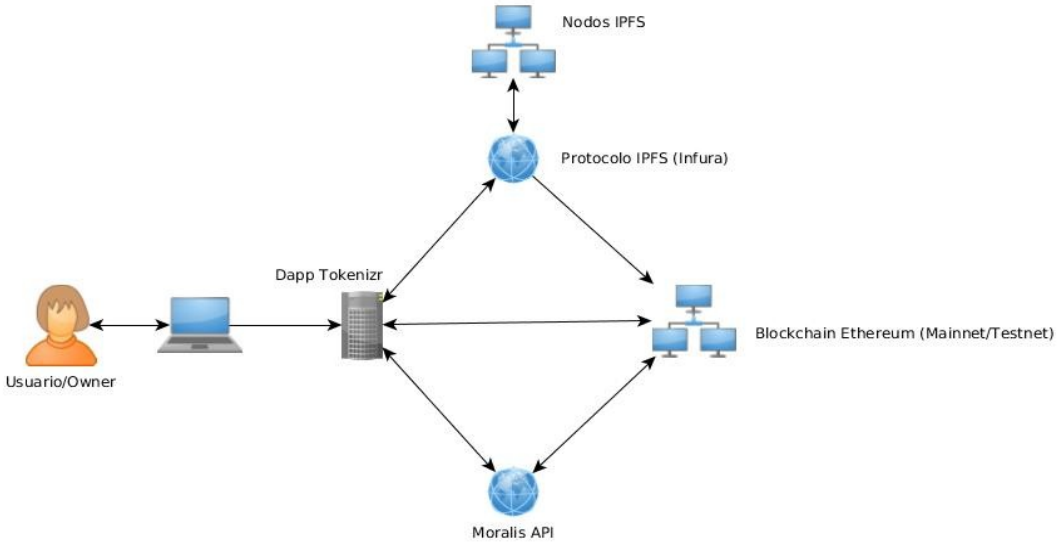


Figura 5: Esquema de conexiones con API

4.3.Análisis de la solución

Para realizar un primer análisis del producto final se realizan una serie de diagramas estándar, que formarán parte del análisis funcional de la aplicación, que unido al libro de fichas de casos de uso compondrían la parte funcional y de toma de requisitos de la aplicación.

4.3.1.Diagrama de casos de uso

Se identifican 5 casos de uso, considerando funcionalidades independientes todas ellas, a excepción del caso de uso de Accout Login, que es el caso de uso que accede a la información de la wallet conectada con la dApp y almacenará una estructura con la información de dicha cartera para ser utilizada en los demás casos de uso.

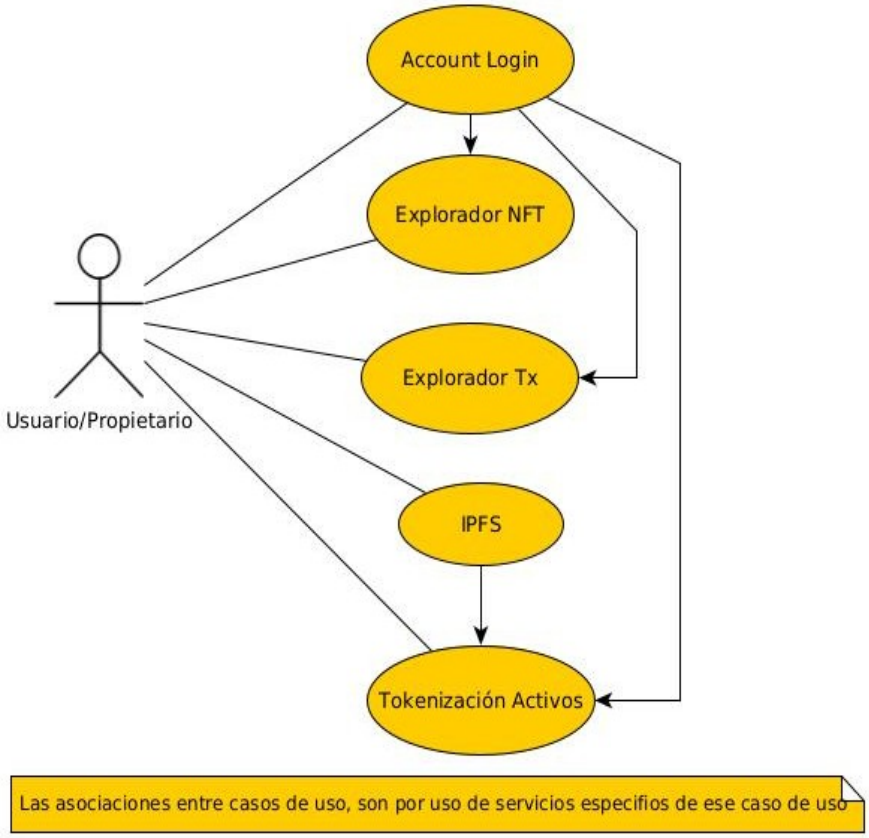


Figura 6: Esquema de casos de uso

4.3.2.Diagrama de componentes

La visión de componentes arroja cuatro componentes agrupados, que se organizarán entorno a las funcionalidades propias de cada una de las pantallas/ casos de uso, y externamente a esas funcionalidades agrupadas se encontrarían componentes específicos para conectar la dApp con nuestra wallet (AccountComponent) y conexiones con IPFS y exploración de bloques (Moralis).

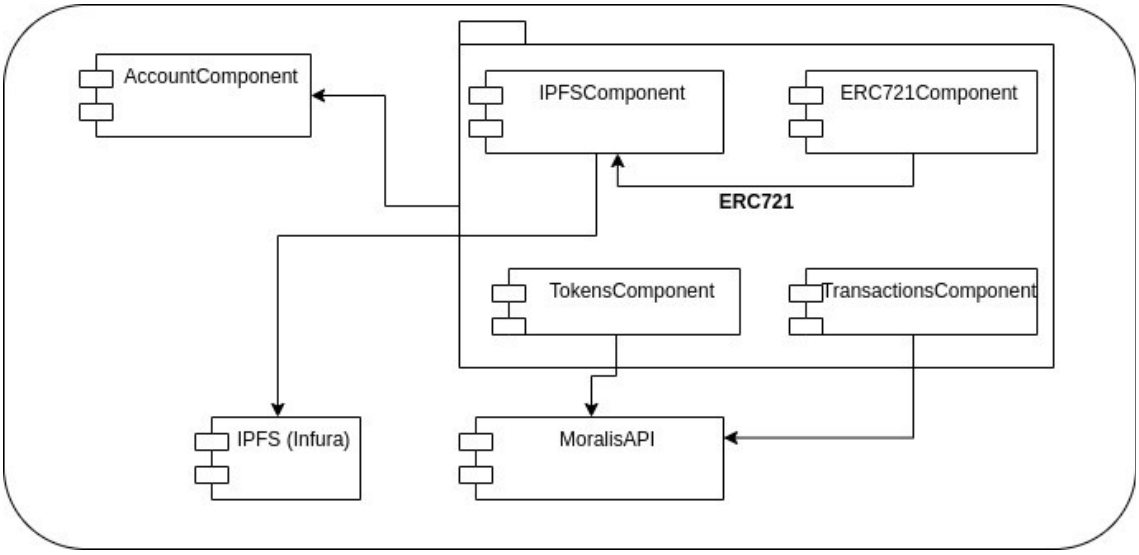


Figura 7: Esquema de componentes

4.3.3.Diagrama de secuencia

Aunque los casos de uso se han diseñado de forma independiente entre sí, hay cierto interdependencia entre ellos, como por ejemplo en los que se muestran en el siguiente diagrama de secuencia, de tal forma que, cuando se despliegue un nuevo token a través del caso de uso ERC721, esto desencadenará que la información mostrada en los casos de uso de Tokens y Transacciones, para la cuenta conectada, mostraría esa nueva información del nuevo contrato desplegado.

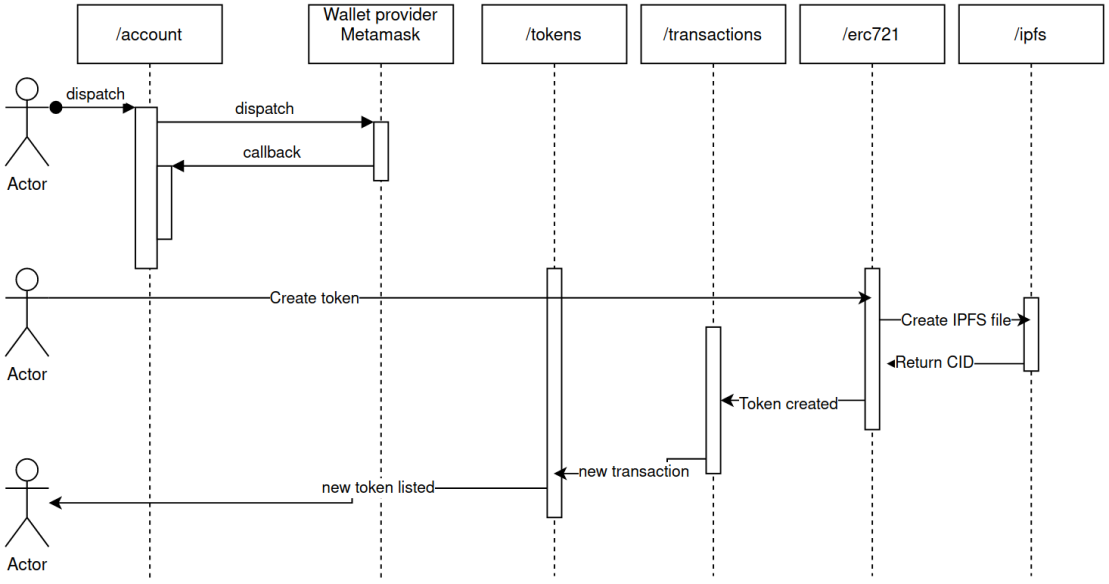


Figura 8: Diagrama de secuencia

4.4.Casos de uso

Se especifican en los puntos siguientes la documentación de cada una de las funcionalidades de la aplicación. Para ello se utilizan fichas de Casos de Uso, que cumplen con el estándar de documentación de Casos de Uso.

4.4.1.- Conexión e información de cuenta

A continuación se especifica la ficha UML para el caso de uso mediante el cual se conectará la dApp a una cuenta configurada en el/los plugins para Wallet del navegador (Metamask/WalletConnect)

Nombre	01.- Conexión de la Dapp con Wallet (Account Login)
Descripción	El usuario debe conectar la aplicación con una cartera a través de una aplicación de Wallet instalada en el navegador, bien sea Metamask o WalletConnect (Ledger Live, Infinity Wallet, Encripted Lnk, Wallet 3,..)
Precondición	El usuario tiene que tener instalado el plugin en el navegador y configurada alguna cuenta en dicho plugin.
Secuencia principal	01 – Instalar plugin de wallet en navegador 02 – Crear o importar cuenta en dicho plugin 03 – Conectar a la Dapp una vez iniciada, seleccionando una de las cuentas configuradas en dicho plugin.
Errores/Alternativas	**No existe plugin metamask instalado:** Utilizar conexión mediante código QR de aplicación móvil. **No hay ninguna cuenta configurada**: Configurar cuenta o importar cuenta.
Post-Condición	Aplicación conectada con una dirección de la Blockchain. La información es visible desde la url /account
Actores	Usuario/Propietario
Notas	En la pantalla se debe especificar la dirección de la cuenta, el balance en ETH, así como el provedor a través del que se ha conectado (Metamask, WalletConnect,..)

Tabla 7: Caso de Uso 01

4.4.2.- Explorador de Tokens NFT/TX

Se especifican a continuación las fichas UML para dos casos de uso para visualizar un pequeño explorador de transacciones y tokens que estén asociados a la cuenta conectada con la aplicación. Las funcionalidades estarán separadas en dos pantallas diferentes.

Nombre	02.- Explorador de Tokens NFT (Explorador NFT)

Descripción	El usuario desea listar los tokens de tipo NFT que están asociados a la cuenta conectada con la aplicación.
Pre-condición	El usuario deberá tener la aplicación conectada con una cuenta de Metamask o WalletConnect, que estén en la red Kovan. Y dicha cuenta tiene ser propietaria de al menos un token NFT, que debe estar importado en la cartera, mediante la dirección del Token.
Secuencia principal	01 Conectar la Dapp mediante Metamask o ConnectWallet 02 Acceder a los detalles de los Tokens NFT de los cuales la wallet es propietaria 03 Interactuar con la tabla
Errores/Alternativas	**No se visualiza la lista**: No existen tokens asociados o no está conectada la aplicación con ninguna wallet. **No se visualiza la lista**: La red configurada no es Kovan, que es la red en la cual se ha configurado la cuenta de prueba de Moralis.
Post-Condición	Se visualizará una tabla con la información de los tokens de tipo ERC721, pertenecientes a la wallet conectada con la Dapp, en la red configurada en ese momento.
Actores	Usuario/propietario, Servidor definido en Moralis sobre Kovan.
Notas	Se deberán mostrar los tokens independientes, aunque se podrá agrupar por token, indicando el numero de tokens que pertenecen al usuario de un determinado contrato.

Tabla 8: Caso de Uso 02

Nombre	03.-Explorador de Transacciones (Explorador Tx)
Descripción	El usuario desea listar las transacciones que están asociadas a la dirección conectada con la aplicación, tanto con dicha dirección como origen como de destino.
Precondición	El usuario deberá tener la aplicación conectada con una cuenta de Metamask o WalletConnect, que estén en la red Kovan. Y dicha cuenta tiene ser propietaria de al menos un token NFT, que debe estar importado en la cartera, mediante la dirección del Token.
Secuencia principal	01 Conectar la Dapp mediante Metamask o ConnectWallet 02 Acceder al listado de las transacciones que se han realizado con la dirección de la wallet conectada con la Dapp. 03 Interactuar con la tabla
Errores/Alternativas	**No se visualiza la lista**: No existen transacciones asociadas a la dirección o no se ha conectado ésta con ninguna wallet. **No se visualiza la lista**: La red configurada no es Kovan, que es la red en la cual se ha configurado la cuenta de prueba de Moralis.
Post-Condición	Se visualizará una tabla con la información de las transacciones que se han

	realizado en la red, pertenecientes a la wallet conectada con la Dapp, en la red configurada en ese momento.
Actores	Usuario/Propietario, Servidor de Moralis sobre Kovan.
Notas	Se deberán mostrar las transacciones con posibilidad de expandir la informacion de cada una de ellas pinchando en su hash, para lo cual se extraerá dicha información de la blockchain utilizando la api de Moralis.

Tabla 9: Caso de Uso 03

4.4.3.- Tokenización de Activos

Se especifica a continuación el caso de uso principal para la tokenización de activos, en el cual partiendo de la estructura de la POC (Prueba de Concepto) entregada en la iteración anterior, se amplia la información recogida y se modifica el contrato para para adaptarlo a toda esta información nueva.

Nombre	04.-Tokenización de activos
Descripción	El usuario desea poder generar Tokens NFT tanto en la mainnet como en las diferentes testnets de Ethereum. Estos Tokens podrán tener asociado uno o varios ficheros o documentos, que deberán estar referenciados mediante CID únicos de la red IPFS. Estos contratos, una vez desplegados, podrán visualizarse en las pantallas de los casos de uso 02 y 03 (Sólo en caso de tesnet Kovan)
Precondición	El usuario deberá tener la aplicación conectada con una cuenta de Metamask o WalletConnect, en principio en cualquier red, tanto mainnet como testnet. Y dicha cuenta tiene ser propietaria de al menos un token NFT, que debe estar importado en la cartera, mediante la dirección del Token. Para poder visualizar los contratos desplegados.
Secuencia principal	01 Conectar la Dapp mediante Metamask o ConnectWallet 02 Rellenar los campos de formulario de información personal del propietario (Owner) del activo tokenizar, que se almacenarán en el contrato. 03 Rellenar los campos de formulario de información del token (nombre, simbolo, supply,...) 04 Rellenar los campos de formulario de información del activo (tipo de activo, bloqueable, ...) 05 Comprobar que se han rellenado todos los campos y desplegar el contrato en la red configurada en la wallet conectada con la aplicación. 06 Esperar la respuesta de confirmación de despliegue y consultar en los casos de uso 02 y 03.
Errores/Alternativas	**No se activa el botón de despliegue**: Se debe asegurar que todos los campos obligatorios en las distintas pantallas están rellenos de una forma

	correcta. **El contrato no se despliega**: Asegurar que la aplicación está conectada a una Wallet, y que esta tiene suficiente gas para realizar el despliegue. **La transacción y los tokens no se muestran en las pantallas de Tx y Tokens:** Asegurar que el despliegue se ha realizado sobre la red en la cual se tiene configurado el servidor de Moralis (Kovan) y que ha transcurrido el tiempo suficiente para que los mineros de la red hayan confirmado la transacción.
Post-Condición	Se desplegará un Token en la red del proveedor de servicios con el cual se encuentra conectada la aplicación, que podrá ser visualizado, tanto la información del token como de la transacción, en las pantallas de los casos de uso 02 y 03.
Actores	Usuario/Propietario
Notas	En este caso de uso se debe realizar una validación de cada uno de los campos para que no se pueden introducir valores extremos que produzcan que no se pueda desplegar el contrato. Se debería introducir una pantalla de verificación o resumen previo al despliegue en la cual se deberá visualizar toda la información que se va a almacenar en el contrato.

Tabla 10: Caso de Uso 04

4.4.4.- Interfaz IPFS

Se especifica a continuación el caso de uso para el desarrollo de una interfaz para despliegue de ficheros a través del protocolo IPFS, que nos permitirá obtener direcciones CID únicas que posteriormente poder asignar dentro de los contratos ERC721.

Nombre	05.- Interfaz de subida de ficheros a IPFS (IPFS)	
Descripción	El usuario desea poder extraer identificadores únicos para ficheros almacenados en la red IPFS (CID), para posteriormente poder asignar dichos identificadores como Base_URI dentro de los contratos ERC721 desplegados.	
Precondición	No es necesaria ninguna condición previa.	
Secuencia principal	01	Seleccionar un fichero de la máquina del usuario.
	02	Incorporar el fichero a la red IPFS a través del servidor de Infura.
	03	Recuperar el CID otorgado por el protocolo y visualizarlo en el listado de CID generados esa sesión de navegador.
Errores/Alternativas	**El fichero no se incorpora a la red**: Comprobar que existe conexión a internet. **El CID generado no se almacena en la sesión**: Comprobar que el JavaScript tiene acceso al localStorage del navegador	
Post-Condición	Se obtendrá un identificador único en la red IPFS por cada fichero que se incorpore a través de la pantalla, y además será posible visualizarlos en un	

	listado de los ficheros incorporados en la sesión de navegador actual.
Actores	Usuario/propietario, Servidor IPFS de Infura.
Notas	Deberá ser posible acceder a los ficheros incorporados a la red en una misma sesión de navegador, por lo que será necesario hacer permanente, para la sesión de navegador en curso, la información (nombre, CID) de los ficheros incorporado.

Tabla 11: Caso de Uso 05

4.4.5.-Modificación/Ampliación del contrato ERC721

Se realiza una ampliación del contrato ERC721, para añadir información del activo, así como otro tipo de información que será de utilidad para los holders/inversores en dicho activo.

Los campos añadidos se han agrupado en diferentes formularios que son los siguientes:

Información del Token:

Se deberá almacenar información referente al token que se generará, que además es obligatoria para el despliegue del contrato.

Nombre campo	Descripción (Tamaño Máx.)
Nombre token	Nombre del token a desplegar (50)
Símbolo token	Símbolo del token (6)
Max Supply	Número máximo de tokens minados
Token URI	Dirección única del token (256)
Descripción	Descripción del token (300)

Tabla 12: Tabla de atributos de información del token

Información del activo:

En este formulario se debe introducir información que especifique algunas normas técnicas para el contrato, y amplí e información del tipo de activo y la forma de invertir en el.

Nombre campo	Descripción (Tamaño Máx.)
Tipo activo	Campo enumerado del tipo de activo que se esta tokenizando.(Inmobiliario, capital riesgo, ...)
Id. del activo	Identificador del activo si existe
Bloqueable	Indica si el token generado será pausable en su minado.
Inversores/holders limitados	Indicador booleano, que indicará si se limita el número de holders del token.

Núm. límite de inversores/holders	En caso de limitación del numero de inversores del token, número máximos de éstos.
Lista abierta de inversores/holders	Indica si los inversores/holders pueden acceder al token de forma abierta o deben ser aprobados previamente por el Owner.

Tabla 13: Información del activo

4.4.6.-Servicios

Se han generado varios servicios que aportan la lógica necesaria para interacturar con los diferentes objetos o entidades. Los servicios que se definen son:

Contract Service: En este servicio se incluirá toda la lógica para interactuar con la red Ethereum, por ejemplo para despliegue de contratos y obtención de información de la cuenta conectada.

IPFS Service: En este servicio se deberá incluir la lógica necesaria para incluir ficheros en la red IPFS a través de su protocolo, ademas también se ofrecerán servicios para obtener un determinado fichero a través de su CID.

ThreeBoxService: Este servicio proporcionará una interfaz para poder conectar la aplicación con un proveedor de servicios de Wallet, como podría ser Metamask o cualquiera de los compatibles con WalletConnect.

4.5.Herramientas y APIs

Para el correcto funcionamiento de la plataforma ha sido necesario la utilización de varias API, entre las que se encuentran la API para conexión al protocolo IPFS y la API de Moralis, que es una plataforma para desarrollo de aplicaciones Web3 descentralizadas, a continuación se especifican las conexiones a dichas API.

4.5.1.Despliegue de nodo IPFS

IPFS son las siglas de “Interplanetary File System”, que en español se traduce como Sistema de Archivos Interplanetario. Más allá de su nombre, esta tecnología se puede describir como un protocolo de almacenamiento descentralizado que permite interacción directa entre sus usuarios por medio de una red P2P global[W-16].

Se ha desplegado un nodo local de IPFS, que permitirá conectarse a la red del protocolo y desplegar ficheros en ella. Para la instalación de este nodo se procede a descargar el cliente[W-17] del sitio web https://ipfs.io, y ejecutarlo en modo stand-alone, para esto se procederá a descargar el paquete “ipfs-desktop-0.17.0-linux-x86_64.appImage”, que una vez descargado en modo local se podrá ejecutar en una consola o terminal con el siguiente comando:

```
$./ipfs-desktop-0.17.0-linux-x86_64.appImage
```

Una vez se inicia, se dispondrá de un nodo conectado a la red IPFS con parámetros de conexión en modo local:

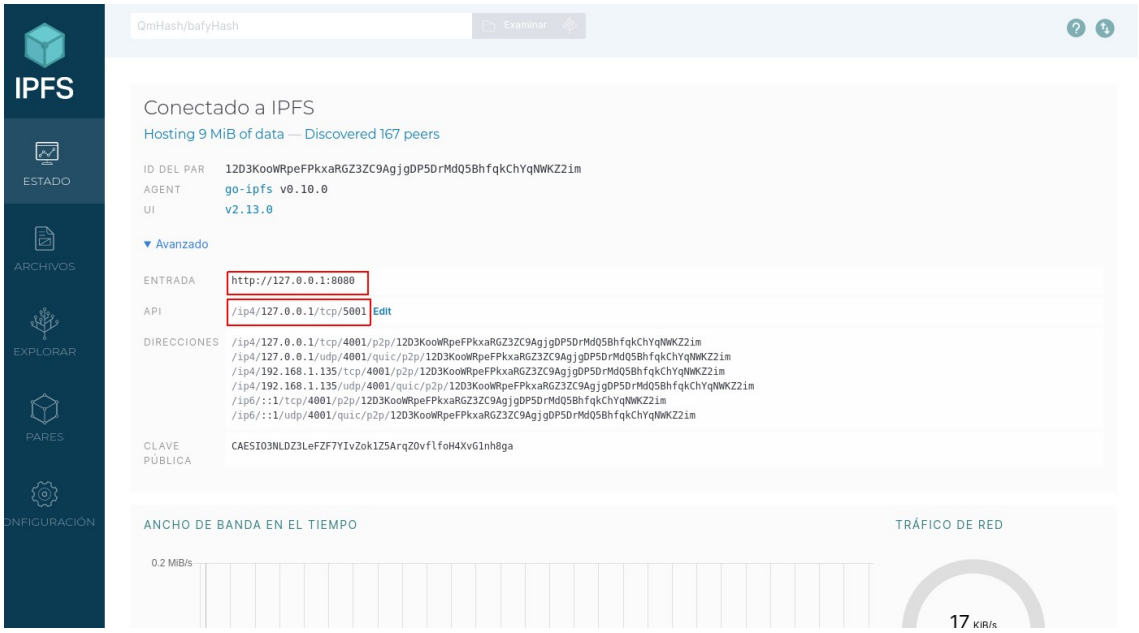


Figura 9: Despliegue de nodo IPFS de manera local

Se puede observar que las URL que ofrece el nodo para conexión, son direcciones locales.

Para el despliegue de ficheros en la red de nodos IPFS se ha optado por instalar el cliente nativo para aplicaciones Angular, mediante la instalación a través del Node Package Manager (npm):

```
$ npm install ipfs-http-client
```

Es posible que se necesite crear un archivo de configuración de WebPack personalizado para decirle a Angular que incluya los módulos de nodo criptográfico en la compilación. Se deberá empezar por instalar el creador de WebPack personalizado, a través de los siguientes comandos:

```
$ npm install -save-dev @angular-builders/custom-webpack
```

4.5.2.Moralis

Para el desarrollo de algunos casos de uso de la aplicación, mas concretamente para el explorador de transacciones de la wallet conectada y

para listar los Tokens NFT que son propiedad de la misma, se hará uso de la API para Web3 de Moralis.

Moralis Web3 API es un método bastante potente para obtener todo tipo de datos de blockchain, como son, por ejemplo, atributos de la cuenta, de los tokens, nativos de la blockchain, etc.

Para el desarrollo de los casos de uso se ha definido un servidor sobre la red de prueba Kovan, con los siguientes parámetros de configuración:

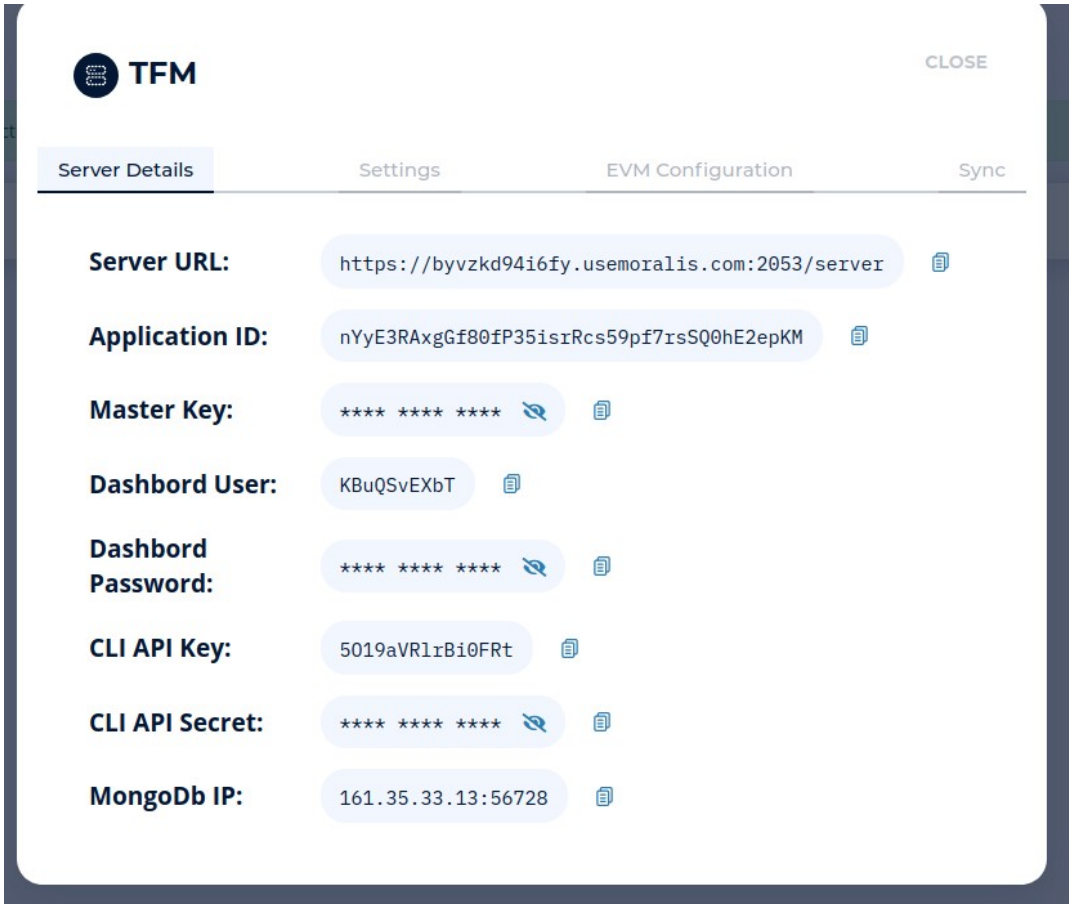


Figura 10: Configuración de servidor en Moralis

Con estos parámetros se puede configurar la aplicación para extraer, a través de una API Web3, todo tipo de información de transacciones, tokens, balances de cuentas, etc.

Para instalar las dependencias en el proyecto, es necesario ejecutar *npm* de la siguiente manera*:*

```
$ npm install moralis
```

Es necesario también configurar el acceso mediante el *appId* y el *server* URL, que se puede definir en el fichero *environment.ts* del proyecto Angular, de la siguiente manera:

```
export const defaultEnv: Env = {
  ...
  moralis: {
    appId: 'MORALIS_APPLICATION_ID',
    serverUrl: 'MORALIS_SERVER_URL'
  }
};
```

Si se desea configurar Moralis para extraer datos de la Mainnet o de otra red de prueba diferente a la red configurada para este TFM, se puede crear una cuenta gratuita en https://admin.moralis.io/login, y posteriormente crear un servidor para cualquiera de las redes tanto mainnet como testnet disponibles:

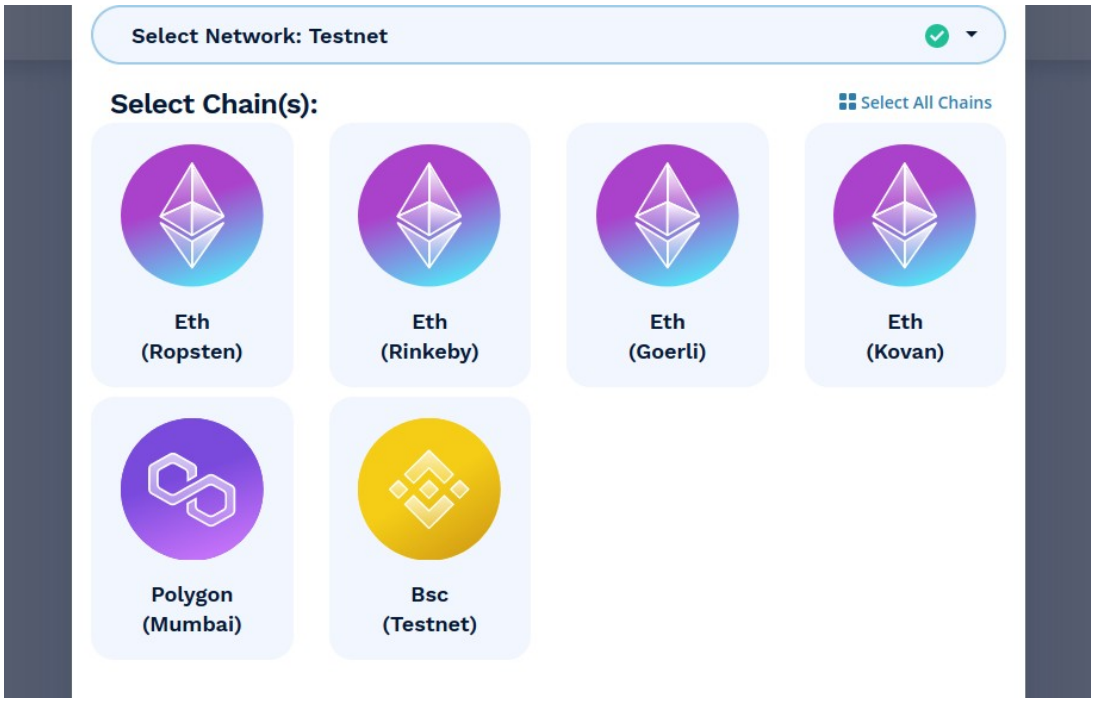


Figura 11: Redes disponibles para API en Moralis

5. Conclusiones y trabajo futuro

5.1.-Conclusiones finales y lecciones aprendidas

A continuación se describen las conclusiones finales del trabajo, y las lecciones aprendidas:

- Durante el desarrollo del proyecto ha quedado demostrada la utilidad de tokenizar un activo, o utilizar tokens para ser usados de muy distintas maneras (otorgar un derecho, distribución de beneficios, método de intercambio,...) en un ámbito de aplicación o empresarial.

- Ha quedado patente que se puede utilizar la tecnología Blockchain en muy diferentes ámbitos y usos, mas allá del uso generalizado y meramente especulativo de las criptomonedas.

- Se ha demostrado, de una manera mas o menos sencilla, que se pueden generar tokens representando a activos del mundo real, y que dichos activos podrán ser participados, transferidos, bloqueados,.. siempre a través de las operaciones que se realicen con sus tokens.

- Se ha demostrado que de una manera sencilla se puede acceder a la blockchain para extraer información relevante de la misma, y que dicha información es pública y transparente sin opción a modificación malintencionada.

- En definitiva se han obtenido conocimientos acerca de la utilidad de la utilización de tecnologías Blockchain en ámbitos distintos a los comúnmente utilizados, y que la tokenización de activos puede aportar valor y gobernanza a la hora de desarrollar cualquier proyecto en ámbitos empresariales, start-ups, etc.

La valoración global es que se han cumplido los objetivos marcados al inicio, aunque debido a la poca experiencia inicial en el tema, el trabajo ha sido duro y continuo, pero en definitiva la recompensa a nivel didáctico ha sido muy positiva.

5.2.-Logro de objetivos iniciales

A continuación se describen una serie de puntos a modo de reflexión crítica sobre el logro de los objetivos planteados inicialmente:

Se puede decir que, en general, se han cumplido todos los objetivos planteados al inicio de la implementación del proyecto con un porcentaje muy alto de desarrollo de todas las unidades de trabajo planteadas inicialmente, aunque en alguno de los sprints si existió un riesgo real de no poder ser

entregado a tiempo debido a falta de experiencia del desarrollador, que aunque si se contempló en el análisis de riesgos inicial del plan de trabajo, tal vez no se pudo cuantificar de forma correcta el impacto de este riesgo en el desarrollo final, ya que cualquier paso a implementar en la plataforma, exigía de una tarea previa bastante costosa de investigación y pruebas de las soluciones a implementar, y no se tuvieron en cuenta esos tiempos en la planificación.

En relación a los objetivos planteados al inicio del proyecto se expone la siguiente valoración de cumplimiento:

Objetivos de investigación/Estudio de Smart Contracts

Aunque siempre es posible seguir profundizando en el tema de aprendizaje acerca del desarrollo de Smart Contracts, implementando nuevos contratos, añadiendo nuevas funcionalidades,etc , se puede considerar que el objetivo de investigación y estudio de la tecnología para desarrollar Smart Contracts, planteado inicialmente en el plan de trabajo e incluido en el Sprint, se ha conseguido en un porcentaje muy alto.

Objetivos de Desarrollo de la plataforma

Se puede concluir que los objetivos planteados al inicio del desarrollo del proyecto, en relación al desarrollo de la plataforma, se han conseguido, en un porcentaje muy alto, añadiendo a lo planteado funcionalmente al inicio, otros casos de uso que ayudan a visualizar o cumplimentar la funcionalidad propia de tokenización.

Objetivos de Presentación Final

Se concluye que en la memoria final se han podido plasmar de forma acertada todos los pasos dados desde su planificación inicial hasta el desarrollo de la plataforma, de tal forma que puedan ser evaluados correctamente.

5.3.-Análisis del seguimiento y planificación

A continuación se expone a modo crítico, en relación al seguimiento de la planificación y metodología, una serie de puntos:

En relación a la planificación realizada y la metodología escogida se puede concluir que, la utilización de una metodología ágil basada en sprints y feedbacks de los mismos, ha contribuido a una mejor planificación de las entregas y un refinamiento progresivo de los productos entregados y, aunque de los productos inicialmente propuestos a incluir o investigar, se han eliminado algunos, como por ejemplo Ethereumjs-tx o MochaJs/ChaiJs, debido principalmente al desconocimiento, se ha implementado en un porcentaje muy alto la planificación inicial.

5.4.-Líneas de trabajo futuro

Una vez conseguidos los objetivos didácticos y de desarrollo que se establecieron al inicio del proyecto se exponen a continuación una serie de líneas de trabajo futuro que mejorarían el proyecto:

- Queda pendiente definir el encaje legal de la tokenización de un activo y establecer quien y como haría efectivos las transferencias, quemados, minado de los activos asociados a los tokens, ya que de momento, aunque toda esta operativa está perfectamente codificada en el Smart Contract, todo lo concerniente al reflejo en el mundo real de las operaciones en la Blockchain, queda fuera del ámbito de este proyecto.

- En relación al punto anterior, una línea de trabajo futuro para la ampliación del proyecto sería el trabajo con Contratos Ricardianos, que aunque a nivel técnico no se diferenciaría mucho de un Smart Contract bajo el estándar ERC721, si llevaría un trabajo paralelo y un acuerdo digital previo, para legalizar todo el proceso de creación de tokens (contratos), transferencia de los mismos, “quemado”, minado etc.

- Aunque la parte de tokenización de los activos es una parte importante en el proceso de gestión de activos mediante tokens, queda patente que existe otra parte, no menos importante, que sería la gestión (transferencia, minado, quemado, aprobación,...) de esos mismos activos y sus tokens, una vez desplegados en la Blockchain, parte esta que ha quedado fuera del ámbito del proyecto. Se propone como línea futura, por lo tanto, la creación de una dApp o ampliación de la existente que sea capaz de operar con los Smart Contracts ERC721, desplegados desde la plataforma actual.

6. Glosario

- **ERC721:**ERC-721 es un estándar abierto y gratuito que describe cómo construir tokens únicos o no fungibles en la cadena de bloques Ethereum.
- **Moralis:** Plataforma para desarrollo de aplicaciones Web3.
- **dApp:**Una dApp dispone de un código backend ejecutándose en una red descentralizada punto a punto.
- **IPFS:** Interplanetary File System, protocolo diseñado para interconectar sistemas a través de P2P
- **Truffle:**Truffle es un entorno de desarrollo, un marco de prueba y una canalización de activos para cadenas de bloques que utilizan la máquina virtual Ethereum (EVM).
- **Web3:**Es una idea para una nueva iteración de la World Wide Web que incorpora la descentralización basada en blockchains.
- **CID:** Content Identifier, identificador de fichero único en una red IPFS.
- **ICO:** Initial Coin Offering, oferta inicial de tokens para financiar proyectos.
- **IDE:** Integrated Development Environment, entorno integral de desarrollo.
- **Ganache**: Blockchain Ethereum local para pruebas.
- **NFT**: No Fungible Token, basado en ERC721.
- **SDK**: Software Development Kit.
- **NPM**: Node Package Management.
- **POC**: Proof of Concept (Prueba de concepto)
- **Contratos Ricardianos**: Es una forma de documentos digitales que actúan como un acuerdo entre dos partes sobre los términos y condiciones para una interacción entre las partes acordadas.

5. Bibliografía

Webs[W]:

[1] https://jfnoriega.com/que-es-la-tokenizacion-y-cuales-son-sus-ventajas/

[2]https://es.beincrypto.com/tokenizacion-del-todo-inmuebles-acciones-vino-mas/

[3] https://ebtc.club/2021/02/24/la-tokenizacion-de-activos-no-esta-despegando-pero-deberia-hacerlo/

[4]https://medium.com/@wmougayar/tokenomics-a-business-guide-to-token-usage-utility-and-value-b19242053416

[5]https://medium.com/@wmougayar/tokenomics-a-business-guide-to-token-usage-utility-and-value-b19242053416

[6]http://startupmanagement.org/2015/02/04/an-operational-framework-for-decentralized-autonomous-organizations/

[7]https://www.larraurimarti.com/es/tokenizacion-de-activos

[8]https://www.legaltoday.com/legaltech/novedades-legaltech/tokenizacion-de-activos-naturaleza-juridica-del-token-y-del-activo-2019-11-20/

[9]https://ethereum.org/en/developers/docs/standards/tokens/erc-721/

[10]https://www.tokenpost.kr/terms/14045

[11]https://infura.io/

[12]http://trufflesuite.com/ganache/

[13]https://infura.io/pricing

[14]http://trufflesuite.com/docs/truffle/

[15]https://github.com/angular-es/AngularTruffleDapp

[16]**h**ttps://www.criptonoticias.com/tecnologia/ipfs-protocolo-almacenamiento-descentralizado-sustituiria-http/

[17]https://docs.ipfs.io/install/ipfs-desktop/

[18]https://docs.openzeppelin.com/contracts/4.x/erc1155

[19]https://101blockchains.com/es/contratos-ricardianos/

Documentos[D]:

[1] Tokenización (PID_00278092): Sistemas de Blockchain. Máster en Ciberseguridad y Privacidad. UOC:

Anexo I
Repositorios de Código

Se habilitan una serie de repositorios de código, en los cuales se han realizado ejemplos, tutoriales, etc, en definitiva se ha realizado el trabajo de investigación inicialmente planteado para el proyecto, y a posteriori ha servido para el desarrollo de la plataforma. También se provee en el mismo repositorio del código fuente de la plataforma.

Repositorio del proyecto:

http://valhons.ddns.net:8081/tfm/final/tokenizr.git

Otros repositorios:

http://valhons.ddns.net:8081/tfm/ejemplos.git http://valhons.ddns.net:8081/tfm/poc1

Credenciales:	
Usuario:	invitado
Password:	1nv1t4d02021

Anexo II
Manual de Instalación

Instrucciones previas

En este anexo se especificarán las acciones a realizar para poder instalar la plataforma desde cero en cualquier sistema. Para ello se indicarán las versiones y software necesario para compilar y ejecutar la plataforma desde cero. Se especificarán las instrucciones de instalación para un sistema Linux basado en Debian, aunque en principio no deberían diferenciarse mucho los comandos para un sistema basado en Windows.

Instalación de Angular/NodeJs/Npm

En primer lugar, y sólo para sistemas que no dispongan de ninguna versión de Angular instalado, para instalar Angular es necesario tener instalado previamente una versión de NodeJs y NPM. Para instalar NodeJs en Ubuntu se deberían ejecutar los siguientes comandos:

```
$ sudo curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash
$ sudo apt install -y nodejs
```

Los comandos anteriores serían efectivos para la instalación de la versión 12, para otra versión bastaría con intercambiar la versión 12.x por la deseada. Pero en principio para el funcionamiento de la plataforma con esta sería suficiente.

Una vez instalado NodeJs, además, para compilar e instalar complementos nativos de NPM, es posible que sea necesario instalar herramientas de desarrollo en su sistema de la siguiente manera:

```
$ sudo apt install -y build-essential
```

Una vez que Node.js y NPM estén instalados, se puede instalar Angular CLI usando el administrador de paquetes npm de la siguiente manera (el indicador -g significa instalar la herramienta en todo el sistema para que la utilicen todos los usuarios del sistema):

```
$ sudo npm install -g @angular/cli
```

Para comprobar que se ha instalado correctamente se podría ejecutar :

```
$ng --version
```

Debiendo obtener una pantalla como la siguiente en caso de que la instalación se haya realizado correctamente:

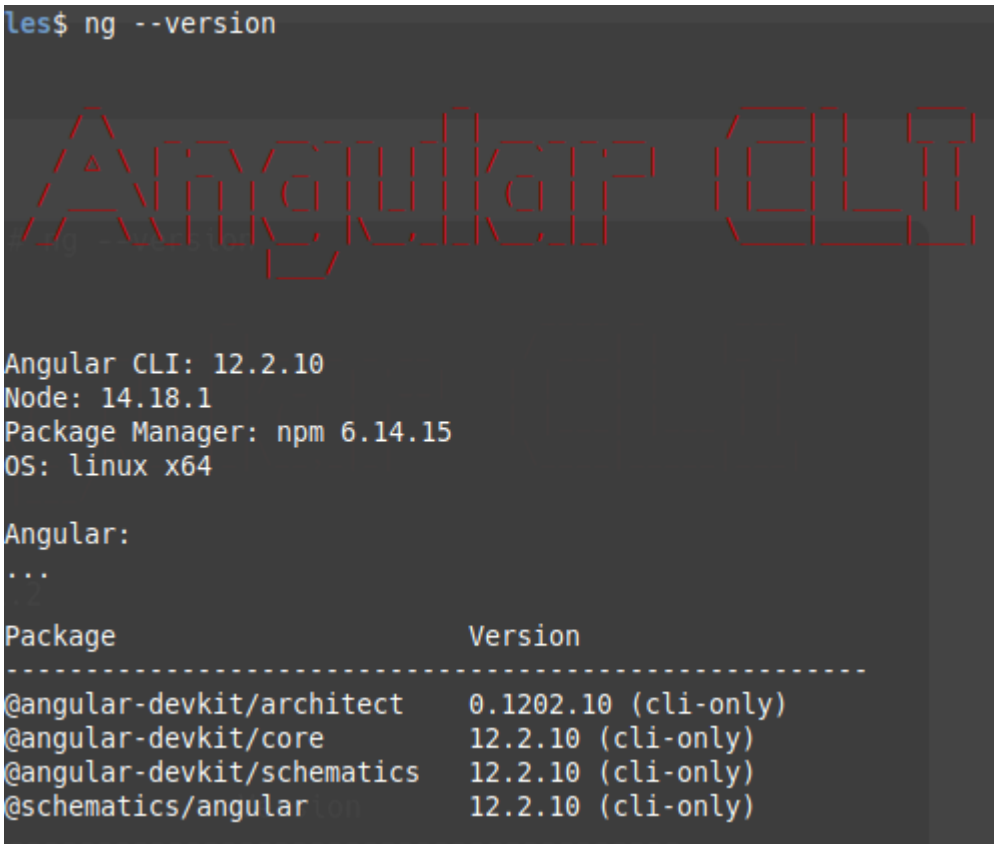


Figura 12: Versión de Angular

Instalación de git

Para realizar el clonado del proyecto es necesario instalar una versión de Git, los pasos para realizarla son los siguientes.

Es necesario previamente en una consola o terminal actualizar las listas de paquetes:

```
$ sudo apt-get update
```

Para instalar de los repositorios por defecto en una consola ejecutar lo siguiente:

```
$ sudo apt-get install git
```

Una vez realizada la instalación, se puede comprobar que se ha instalado el software ejecutando desde una consola el siguiente comando:

```
$ git --version
```

Obteniendo la versión instalada:

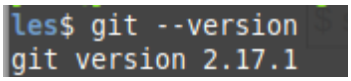


Figura 13: Versión de Git

Instrucciones para clonar el repositorio, instalar y desplegar:

Previo a la descarga del proyecto, es necesario asegurar que el core de Angular, así como la librería de Material y el cliente se encuentran actualizados en la máquina destino, para ello se deberán ejecutar los siguientes comandos en un terminal:

```
$ ng update @angular/cli @angular/core @angular/material
```

Para clonar el repositorio y ejecutar la aplicación en un servidor NodeJS, es necesario seguir las siguientes instrucciones:

Clonar el repositorio en un directorio local, mediante el siguiente comando:

```
$ git clone http://valhons.ddns.net:8081/tfm/final/tokenizr.git
```

Una vez clonado el repositorio es necesario acceder al directorio “tokenizr” :

```
$ cd tokenizr
```

Una vez en dicho directorio ejecutar el comando npm para instalar dependencias:

```
$ npm install
```

Este proceso puede tardar unos minutos dependiendo de la cantidad de dependencias que ya existan en el equipo y de la velocidad de la conexión a internet.

Una vez finalizado el proceso de instalación de dependencias, se puede ejecutar el servidor NodeJS para levantar la aplicación mediante el siguiente comando:

```
$ ng serve
```

Una vez arrancado el servidor se puede acceder a la aplicación, a través de la dirección *http://localhost:4200*, en un navegador compatible. Es recomendable tener instalado el plugin de Metamask en el navegador, para poder realizar la conexión con la aplicación.

Instalación de dependencias IPFS

Para el despliegue de ficheros en la red de nodos IPFS es necesario instalar el cliente nativo para aplicaciones Angular, mediante la instalación a través del Node Package Manager (npm):

```
$ npm install ipfs-http-client
```

Es posible que se necesite crear un archivo de configuración de Webpack personalizado para decirle a Angular que incluya los módulos de nodo criptográfico en la compilación. Se deberá empezar por instalar el creador de WebPack personalizado, a través de los siguientes comandos:

```
$ npm install -save-dev @angular-builders/custom-webpack
```

Instalación de dependencias de Moralis

Para instalar las dependencias en el proyecto, es necesario ejecutar npm de la siguiente manera:

```
$ npm install moralis
```

Es necesario tambien configurar el acceso mediante el appId y el server Url, que se puede definir en el fichero environment.ts del proyecto Angular, de la siguiente manera:

```
export const defaultEnv: Env = {
        ...
        moralis: {
                appId: 'MORALIS_APPLICATION_ID',
                serverUrl: 'MORALIS_SERVER_URL'
        }
};
```