



Escola Politécnica Superior
d'Enginyeria de Vilanova i la Geltrú

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO: DApp de compraventa y alquiler de inmuebles

AUTOR: Montesinos Parra, Héctor

FECHA: Febrero, 2022

APELLIDOS:	MONTESINOS PARRA	NOMBRE:	HÉCTOR
TITULACIÓN:	GRADO EN INGENIERÍA INFORMÁTICA		
PLAN:	2018		
DIRECTOR:	EVA MARÍN TORDERA		
DEPARTAMENTO:	ARQUITECTURA DE COMPUTADORES		

CALIFICACIÓN DEL TFG

TRIBUNAL

PRESIDENTA
FARRERAS ESCLUSA,
MONTSERRAT

SECRETARIO
ACOSTA COBOS,
MARIO CESAR

VOCAL
FERNÁNDEZ
GONZÁLEZ, JULIO

FECHA DE LECTURA: 08/02/2022

Este Proyecto tiene en cuenta aspectos medioambientales: Sí No

RESUMEN

Históricamente, el sector inmobiliario se ha visto afectado por la complejidad de sus procesos, sumado a una falta de transparencia, debido a un alto número de documentos e intermediarios participantes en dichos procesos.

Actualmente, la rompedora tecnología Blockchain, se abre camino en diferentes sectores, debido a ciertos factores que aporta, como transparencia, inmutabilidad y trazabilidad en las transacciones.

En el presente proyecto se pretende crear una plataforma web descentralizada, lo que se conoce como una DApp (Decentralized Application) en un entorno blockchain, la cual permita agilizar procesos burocráticos, eliminando intermediarios, tiempos de espera y costes de gestión asociados a la compraventa o alquiler, además de acercar a los usuarios a las aplicaciones descentralizadas, brindándoles una buena experiencia de usuario.

Para llevar a cabo esta prueba de concepto, se realizará una investigación dentro del campo de la tecnología Blockchain, estudiando todo su potencial y características, dónde se utilizará un stack de herramientas y tecnologías que actualmente están a la vanguardia, detallando cada una de ellas, para posteriormente, llevar a cabo la implementación de la DApp.

Palabras clave (máximo 10):

Cadena de bloques	Descentralización	Contrato inteligente	Tokenización
Plataforma	Web		

RESUM

Històricament, el sector immobiliari s'ha vist afectat per la complexitat dels seus processos, afegit a una falta de transparència, a causa d'un gran nombre de documents i intermediaris participants en aquests processos.

Actualment, la trencadora tecnologia Blockchain, s'obre camí en diferents sectors, gràcies a certs factors que aporta, com transparència, immutabilitat i traçabilitat a les transaccions.

Al present projecte es pretén crear una plataforma web descentralitzada, el que es coneix com una DApp (Decentralized Application) en un entorn blockchain, que permeti agilitzar processos burocràtics, eliminant intermediaris, temps d'espera i costos de gestió associats a la compravenda o lloguer, a més d'apropar als usuaris a les aplicacions descentralitzades, brindant-los una bona experiència d'usuari.

Per a dur a terme aquesta prova de concepte, es realitzarà una investigació dins del camp de la tecnologia Blockchain, estudiant tot el seu potencial i característiques, on s'utilitzarà un stack d'eines i tecnologies que en l'actualitat estan a l'avantguarda, detallant cadascuna d'elles, per a, posteriorment, dur a terme la implementació de la DApp.

Paraules clau (màxim 10):

Cadena de blocs	Descentralització	Contracte intel·ligent	Tokenització
Plataforma	Web		

ABSTRACT

Historically, the real estate sector has been affected by the complexity of its processes, added to a lack of transparency, due to a large number of documents and intermediaries participating in these processes.

Currently, the breaking Blockchain technology opens its way in different sectors, thanks to certain factors it brings, such as transparency, immutability and traceability to transactions.

This project aims to create a decentralized web platform, known as a DApp (Decentralized Application) in a blockchain environment, which allows for streamlining bureaucratic processes, removing intermediaries, waiting times and management costs associated with the sale or rental, in addition to bringing users closer to decentralised applications, providing them with good user experience.

To carry out this concept test, an investigation will be carried out within the field of Blockchain technology, studying all its potential and characteristics, where a stack of tools and technologies that are currently in the vanguard will be used, detailing each of them, to then carry out the implementation of the DApp.

Keywords (10 maximum):

Blockchain	Decentralization	Smart contract	Tokenization
Platform	Web		

ÍNDICE

1. INTRODUCCIÓN	11
1.1 CONTEXTO	11
1.1.1 INTRODUCCIÓN	11
1.1.2 CONCEPTOS	11
1.1.3 PROBLEMA QUE RESUELVE	13
1.2 PLANIFICACIÓN	14
1.3 POSIBLES DESVIACIONES	14
1.4 DIAGRAMAS DE GANTT	15
2. ESPECIFICACIÓN Y DISEÑO	15
2.1 ACTORES	15
2.2 ESPECIFICACIÓN	16
2.3 ARQUITECTURA	17
2.4 CASOS DE USO	18
3. TECNOLOGÍAS DESCENTRALIZADAS	37
3.1 BLOCKCHAIN	37
3.1.1 CARACTERÍSTICAS	37
3.1.2 TIPOS DE BLOCKCHAIN	37
3.1.3 ETHEREUM	40
3.1.3.1 BLOQUES	40
3.1.3.2 TIPOS DE CUENTA	41
3.1.3.3 TOKEN ETH Y SUS ESTÁNDARES	42
3.1.3.4 ETHEREUM VIRTUAL MACHINE (EVM)	42
3.1.3.5 GAS	44
3.2 IPFS	45
4. IMPLEMENTACIÓN	47
4.1 BLOCKCHAIN	48
4.1.1 SMART CONTRACTS	48
4.1.2 TRUFFLE	49
4.1.3 GANACHE	51
4.1.4 ORÁCULOS Y REDES	55
4.1.5 TOKENIZACIÓN	60
4.2 METAMASK	62
4.3 FRONT-END WEB	66
4.3.1 VUE.JS	66
4.3.2 NUXT.JS	67
4.3.3 NODE.JS	67
4.3.4 WEB3.JS	67
4.4 IPFS	68
4.5 DATOS EXTERNOS	69
4.6 ESTRUCTURA	71

4.7 PANTALLAS Y NAVEGACIÓN	73
5. CONCLUSIONES	74
5.1 CUMPLIMIENTO DE OBJETIVOS	74
5.2 CONCLUSIONES PERSONALES	74
5.3 FUTURAS MEJORAS	75
6. AGRADECIMIENTOS	77
7. BIBLIOGRAFÍA	78
ANEXOS	80
ANEXO 1: SMART CONTRACTS	80
SMART CONTRACT AUTH	80
INTRODUCCIÓN Y ESTRUCTURAS DE DATOS	80
EVENTOS	81
FUNCIONES	81
SMART CONTRACT PROPERTIES	84
INTRODUCCIÓN Y ESTRUCTURAS DE DATOS	84
EVENTOS	87
FUNCIONES	87
ANEXO 2: PANTALLAS DE NAVEGACIÓN	91
ANEXO 3: DIAGRAMAS DE GANTT	102

ÍNDICE DE FIGURAS

1. RED DESCENTRALIZADA	11
2. TRANSACCIONES ALMACENADAS EN UN BLOQUE	12
3. COMPARATIVA CON EL PROCESO DE COMPROVVENTA ACTUAL	13
4. ARQUITECTURA DE LA DAPP	17
5. DIAGRAMA DE SECUENCIA CASO DE USO "CONECTAR BILLETERA"	19
6. DIAGRAMA DE SECUENCIA CASO DE USO "REGISTRAR USUARIO"	21
7. DIAGRAMA DE SECUENCIA CASO DE USO "LOGIN USUARIO"	23
8. DIAGRAMA DE SECUENCIA CASO DE USO "CONSULTAR PERFIL USUARIO"	24
9. DIAGRAMA DE SECUENCIA CASO DE USO "LOGOUT USUARIO"	26
10. DIAGRAMA DE SECUENCIA CASO DE USO "PUBLICAR PROPIEDAD"	28
11. DIAGRAMA DE SECUENCIA CASO DE USO "COMPRAR PROPIEDAD"	30
12. DIAGRAMA DE SECUENCIA CASO DE USO "ALQUILAR PROPIEDAD"	32
13. DIAGRAMA DE SECUENCIA CASO DE USO "COMPRAR TOKENS"	34
14. DIAGRAMA DE SECUENCIA CASO DE USO "ELIMINAR PROPIEDAD"	36
15. CREACIÓN DE UNA CUENTA MEDIANTE LA LIBRERÍA PERSONAL_NEWACCOUNT	41
16. ARQUITECTURA DE LA EVM	43
17. EJEMPLO DE BYTECODE	43
18. EJEMPLO DE ABI DERIVADO DEL BYTECODE DE LA "FIGURA 17"	44
19. COMPARACIÓN ENTRE ARQUITECTURAS HTTP E IPFS	45
20. SUBIDA DE UN FICHERO A IPFS Y POSTERIOR VISUALIZACIÓN	46
21. VISUALIZACIÓN DE FICHERO SUBIDO A IPFS, FILTRADO MEDIANTE SU HASH CID	46
22. VISUALIZACIÓN DE FICHERO SUBIDO A IPFS MEDIANTE URL	46
23. ARQUITECTURA SEGÚN LAS TECNOLOGÍAS UTILIZADAS	47
24. CONFIGURACIÓN RED DE DESARROLLO EN TRUFFLE-CONFIG.JS	49
25. CONFIGURACIÓN FICHERO DE MIGRACIONES DE TRUFFLE	49
26. IMPLEMENTACIÓN DE TEST EN JAVASCRIPT	50
27. CONFIGURACIÓN DE GANACHE	51
28. ENTORNO GRÁFICO DE GANACHE	52
29. VISUALIZACIÓN DE BLOQUES MINADOS EN GANACHE	53
30. VISUALIZACIÓN DE TRANSACCIONES EN GANACHE	53
31. VISUALIZACIÓN DETALLE DE TRANSACCIÓN EN GANACHE	54
32. VISUALIZACIÓN DE CONTRATO INTELIGENTE DESDE GANACHE	54
33. INTERFAZ DE INFURA	56
34. CONFIGURACIÓN DE RED KOVAN EN TRUFFLE-CONFIG.JS	57
35. IMPLEMENTACIÓN SMART CONTRACT PRICECONVERTER.SOL	58
36. CONVERSIÓN EUR A ETH DESDE LA CONSOLA DE TRUFFLE	59
37. OBTENCIÓN BLOQUE ACTUAL DESDE LA CONSOLA DE TRUFFLE	59
38. OBTENCIÓN DE FECHA MEDIANTE TIMESTAMP EN CONSOLA DE LINUX	59
39. CONVERSIÓN EUR A ETH DESDE CONVERSOR WEB DE CRIPTODIVISAS	60
40. IMPLEMENTACIÓN SMART CONTRACT MINTNFT.SOL	61
41. CONTENIDO DEL FICHERO PASADO COMO ARGUMENTO EN LA FUNCIÓN MINTNFT	61
42. GENERACIÓN DE NFT DESDE LA CONSOLA DE TRUFFLE	61
43. TRANSACCIÓN GENERADA AL EJECUTAR LA FUNCIÓN MINT EN LA RED DE KOVAN VISTA EN ETHERSCAN	62
44. PROCESO DE CONFIGURACIÓN RED DE DESARROLLO EN METAMASK	63
45. VISUALIZACIÓN DE BILLETERA GANACHE EN METAMASK EN RED DE DESARROLLO	63
46. CAMBIO DE RED EN METAMASK Y VISUALIZACIÓN DE LA NUEVA RED CONFIGURADA	64
47. DETALLE DE UNA TRANSACCIÓN EN METAMASK	64
48. IMPORTACIÓN DE TOKEN ERC-721 EN METAMASK	65
49. VISUALIZACIÓN DE TOKEN ERC-721 EN METAMASK	65
50. ARQUITECTURA MVVM	68
51. COMUNICACIÓN MEDIANTE WEB3.JS CON LA BLOCKCHAIN	68
52. CONEXIÓN CON SMART CONTRACTS DESDE NODE.JS	68
53. SUBIDA DE IMÁGENES A IPFS DESDE COMPONENTE VUE.JS	68
54. LECTURA DE DATOS REMOTOS A TRAVÉS DE AXIOS	69

55. INTERFAZ DE STRAPI	69
56. ESTRUCTURA DE FICHEROS DEL PROYECTO	71
57. DIAGRAMA DE PANTALLAS DE NAVEGACIÓN	73
58. ESTADO FINAL DEL KANBAN PERSONAL	75
59. CABECERA Y ESTRUCTURAS DE DATOS SMART CONTRACT AUTH	80
60. EVENTOS SMART CONTRACT AUTH	81
61. FUNCIÓN SIGNUP SMART CONTRACT AUTH	81
62. FUNCIÓN SIGNIN SMART CONTRACT AUTH	82
63. FUNCIÓN GETUSER SMART CONTRACT AUTH	82
64. FUNCIÓN GETADDRFROMINDEX SMART CONTRACT AUTH	82
65. FUNCIÓN CHECKIFUSERLOGGEDBYADDR SMART CONTRACT AUTH	82
66. FUNCIÓN LOGOUTBYADDR SMART CONTRACT AUTH	83
67. HEADER Y ESTRUCTURAS DE DATOS SMART CONTRACT PROPERTIES - PARTE I	84
68. ESTRUCTURAS DE DATOS SMART CONTRACT PROPERTIES - PARTE II	86
69. EVENTOS SMART CONTRACT PROPERTIES	87
70. FUNCIÓN UPLOADPROPERTY SMART CONTRACT PROPERTIES	87
71. FUNCIÓN ADDPROPERTYDATA SMART CONTRACT PROPERTIES	88
72. FUNCIÓN SENDBALANCE SMART CONTRACT PROPERTIES	88
73. FUNCIÓN BUYPROPERTY SMART CONTRACT PROPERTIES	88
74. FUNCIÓN RENTPROPERTY SMART CONTRACT PROPERTIES	89
75. FUNCIÓN BUYTOKENS SMART CONTRACT PROPERTIES	89
76. FUNCIÓN REMOVEPROPERTY SMART CONTRACT PROPERTIES	89
77. FUNCIÓN PROPERTYSETTLED SMART CONTRACT PROPERTIES	90
78. FUNCIÓN GETPROPERTYBYID SMART CONTRACT PROPERTIES	90
79. FUNCIÓN GETPROPERTYOWNER SMART CONTRACT PROPERTIES	90
80. PANTALLA DE INICIO	91
81. PANTALLA DE REGISTRO	92
82. PROCESO DE CONEXIÓN CON LA BILLETERA METAMASK	92
83. MODAL DE CONEXIÓN CON LA BILLETERA METAMASK	92
84. PANTALLA DE LOGIN	93
85. MODAL DETALLES DE USUARIO	93
86. PANTALLA PUBLICACIÓN DE PROPIEDADES - PARTE I	94
87. PANTALLA PUBLICACIÓN DE PROPIEDADES - PARTE II	95
88. PANTALLA PUBLICACIÓN DE PROPIEDADES - PARTE III	95
89. MODAL DE RESTRICCIÓN I EN PANTALLA DE PUBLICACIÓN	96
90. MODAL DE RESTRICCIÓN II EN PANTALLA DE PUBLICACIÓN	96
91. MODAL DE APROBACIÓN EN PANTALLA DE PUBLICACIÓN	96
92. MODAL DE CONFIRMACIÓN EN PANTALLA DE PUBLICACIÓN	96
93. PANTALLA DE PROPIEDADES	97
94. MODAL DE PROPIEDAD EN VENTA	98
95. MODAL DE PROPIEDAD EN ALQUILER	98
96. MODAL DE PROPIEDAD EN ALQUILER POR TOKENS	98
97. OPCIÓN DE ELIMINAR PROPIEDAD DESDE MODAL	99
98. MODAL DE APROBACIÓN DE TRANSACCIÓN	99
99. MODAL DE CONFIRMACIÓN DE TRANSACCIÓN	100
100. MODAL CON INFORMACIÓN DEL CONTRATO GENERADO	100
101. VISUALIZACIÓN DE CONTRATO GENERADO EN IPFS	101
102. DIAGRAMA DE GANTT INICIAL	102
103. DIAGRAMA DE GANTT FINAL	103

GLOSARIO DE SIGNOS, SÍMBOLOS, ABREVIATURAS, ACRÓNIMOS Y TÉRMINOS

DApp: *Decentralized Application*

P2P: *Peer-to-peer*

IPFS: *InterPlanetary FileSystem*

ETH: *Símbolo del token Ether*

API: *Application Programming Interface*

CID: *Content identifier*

ERC: *Ethereum Request for Comments*

NFT: *Non-Fungible Token*

1. INTRODUCCIÓN

1.1 CONTEXTO

1.1.1 INTRODUCCIÓN

El presente proyecto ha sido realizado como trabajo de fin de estudios en el grado de ingeniería informática cursado en la Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú (EPSEVG) de la Universitat Politècnica de Catalunya (UPC).

El proyecto consiste en el desarrollo de una plataforma web dedicada a la compraventa y alquiler de inmuebles, de manera totalmente descentralizada, gracias a la tecnología Blockchain, creando así lo que actualmente se conoce como una DApp (Decentralized Application).

1.1.2 CONCEPTOS

- **Hashing:** Transformación de datos de entrada de longitud variable a una cadena de tamaño fijo, realizada mediante un algoritmo criptográfico específico. Las funciones de hash son unidireccionales, lo que significa que una vez se obtiene la respuesta criptográfica a una entrada de datos, no es posible recuperar la entrada de datos original mediante la respuesta.
- **Blockchain:** El funcionamiento de una cadena de bloques, se podría comparar análogamente a un libro de contabilidad o una base de datos pública, que almacena un historial de transacciones entre las partes implicadas, en conjuntos de datos llamados bloques.

Los usuarios que participan en la blockchain, se denominan nodos, los cuales forman una red P2P, dónde todos los nodos son iguales en cuanto a jerarquía, actuando como cliente y servidor.

Por tanto, los bloques de la cadena, se almacenan formando un sistema descentralizado de manera pública, de forma que cualquiera puede acceder a ellos.

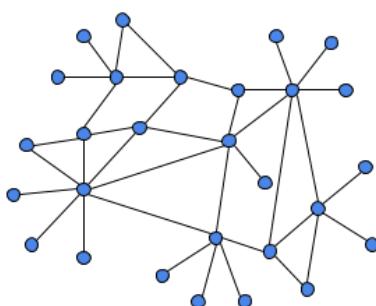


Figura 1: Red descentralizada
Fuente: Elaboración propia

Existen algunas diferencias respecto a las bases de datos que conocemos, las cuales son centralizadas.

La primera de ellas es, que en blockchain, sólo es posible agregar datos, una vez se han escrito los mismos, no se pueden modificar (siguiendo el principio de inmutabilidad) y esa información estará guardada en la cadena de bloques para siempre.

La segunda diferencia es que, blockchain está formada por una lista enlazada de bloques, en lugar de por una tabla, dónde cada bloque que se agrega está enlazado al bloque anterior, mediante un número de hash.

El hash del bloque, está generado por los lotes de transacciones que contiene dicho bloque, de esta forma, cada bloque contiene su propio hash, y el hash del bloque anterior. Esto previene posibles fraudes, ya que cualquier cambio en cualquier bloque alteraría los datos de toda la cadena e invalidaría los bloques posteriores.

A colación con el punto anterior, y como última diferencia, ya que las cadenas de bloques están descentralizadas, no son propiedad de ninguna organización y no se pueden eliminar, a diferencia de una base de datos convencional. Este último punto se debe a que, una cadena de bloques se copia entre todos los nodos de la red.

Cada nodo contiene una copia exacta de todo el historial de la cadena de bloques y, por tanto, dado que todos estos nodos se comunican entre sí y acuerdan el estado de la cadena de bloques constantemente, la única forma de modificar datos ya existentes dentro de la cadena, sería convenciendo al 51% de los nodos que participan en ella.

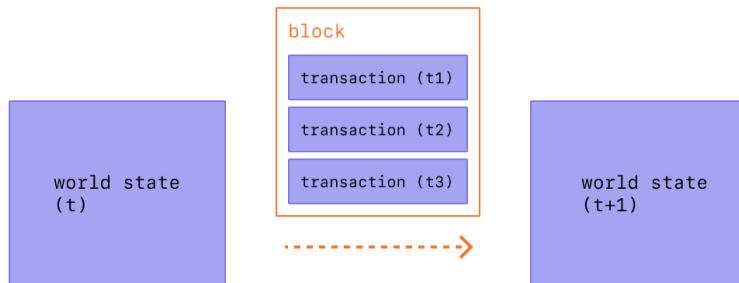


Figura 2: Transacciones almacenadas en un bloque
Fuente: ethereum.org

- **Smart contract:** Los contratos inteligentes son programas autónomos que definen un acuerdo público. Los términos del acuerdo son ejecutados cuando el programa es llamado por un usuario participante en la cadena de bloques. Son desarrollados mediante lenguajes de programación específicos para ello.
- **Oráculo:** Es un puente entre la blockchain y el mundo real. Actúa como servicio interno dentro de la cadena de bloques, lo que se denomina como

entorno on-chain, con el objetivo de aportar información del mundo exterior a un smart contract. Es necesario debido a que, la blockchain y sus smart contracts están limitados a sólo obtener información y acceder a datos que estén dentro de su propia red P2P.

- **Tokenización:** Proceso por el cual se genera un token en un smart contract, el cual representa y da valor a un activo real.

1.1.3 PROBLEMA QUE RESUELVE

Actualmente los procesos que se llevan a cabo en el mercado inmobiliario para adquirir o alquilar inmuebles están plagados de inconvenientes, entre los que se pueden destacar la falta de verificación de la propiedad, lentitud en los procesos burocráticos con un gran número de intermediarios participando en dichos procesos, pagos de altas cantidades en costes notariales, alta probabilidad de existencia de errores humanos o la posibilidad de manipulación de transacciones.

Además de todo lo anteriormente mencionado, las plataformas dedicadas a la búsqueda de inmuebles, están faltas de procesos estandarizados, y en muchas ocasiones pueden provocar tener publicados datos inexactos, incompletos o desactualizados respecto a otras plataformas del mismo sector.

Mediante el uso de la blockchain y los smart contracts, sería posible registrar los términos y condiciones de un contrato de compraventa o alquiler, dentro de un smart contract, el cual se firmaría criptográficamente por las partes que intervienen en el mismo, y quedaría almacenado en la blockchain.

Gracias a la tecnología blockchain y a sus características, las cuáles se especificarán más adelante, las partes serían capaces de realizar una transacción sin necesidad de posibles manipulaciones por parte de terceros y de manera confiable entre ellos, sin necesidad de conocer los datos personales de la otra parte, además de ahorrar tiempos de espera y costes de gestión, notaría, etc.

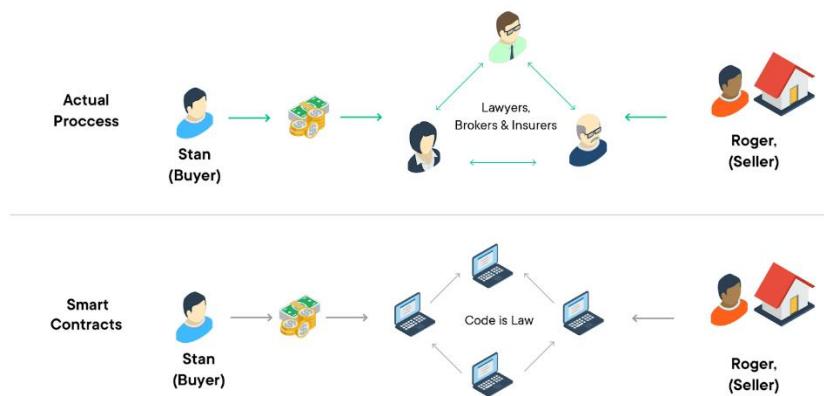


Figura 3: Comparativa con el proceso de compraventa actual
Fuente: feed.swissborg.com

1.2 PLANIFICACIÓN

El presente proyecto tiene una duración aproximada de 4 meses desde el momento de la redacción de la descripción y su posterior matriculación, con fecha de inicio a 22 de septiembre de 2021 y plazo máximo de entrega a 31 de enero de 2022.

La estimación inicial en cuanto a carga de trabajo, es aproximadamente de unas 450 horas, dentro de las cuales están incluidas todas las tareas referentes al proyecto.

La planificación estará dividida en cuatro bloques principales:

- **Estudio y gestión:** Incluye todo el trabajo transparente que hay detrás del proyecto, incluyendo toda la gestión del mismo, reuniones con la directora, seguimiento de metodologías ágiles y organización semanal mediante un KanBan, dónde en nuestro caso usaremos Trello [1], y estudio de las tecnologías y herramientas a utilizar.
- **Especificación y diseño:** Planteamiento de las funcionalidades de las que dispondrá el proyecto, detallando casos de uso, actores implicados y una visión general de la aplicación.
- **Documentación:** Cómo su nombre indica, hace referencia a todo el material redactado para mostrarse en la memoria final. Este bloque irá avanzando simultáneamente junto al bloque de implementación.
- **Implementación:** Desarrollo de código realizada en base al bloque de especificación y diseño, cualquier cambio en dicho bloque afectará a la implementación de la aplicación, para llevar un control del código y sus versiones se utilizará Git [2].

1.3 POSIBLES DESVIACIONES

Teniendo en cuenta que el proyecto se matricula el 22 de septiembre y que la fecha límite de entrega es el 31 de enero, se disponen de algo más de 18 semanas para su planteamiento y desarrollo.

Podría haber razones que provoquen una desviación temporal en la planificación inicial:

- **Uso de nuevas tecnologías en auge:** El alumno no tiene conocimientos de ellas hasta la fecha, por lo que deberá dedicar tiempo a aprenderlas.
- **Mala planificación en la gestión de tareas:** Es posible que, debido al punto anterior, la previsión de tiempos pueda ser poco precisa en algunas tareas de desarrollo, que provoquen que algunas tareas sobrepasen el tiempo inicialmente estipulado, y otras finalicen antes de tiempo.
- **Contratiempos en términos de implementación:** Puede suceder que aparezcan escenarios no contemplados durante la planificación que dificulten o imposibiliten llevar a cabo ciertas acciones.

Por otro lado, dentro de la planificación inicial, si una tarea conllevase más del doble del tiempo inicialmente planificado para que sea completada, se buscaría una alternativa viable a la misma, para asegurar que el proyecto se finalice dentro de los límites estipulados.

1.4 DIAGRAMAS DE GANTT

Tras planificar el proyecto, se ha realizado un diagrama de Gantt con la planificación inicial, y al finalizar el proyecto, se realizará otro diagrama final para poder comprobar las diferencias entre ellos y, en caso de que haya una gran diferencia, aprender de los errores de cara a próximos proyectos.

Ambos diagramas se adjuntan en el Anexo 3 del presente documento.

2. ESPECIFICACIÓN Y DISEÑO

2.1 ACTORES

Las partes implicadas en una DApp, son básicamente los usuarios que la utilizan, y la propia aplicación. Como ya se ha comentado en el apartado de introducción, una blockchain no es propiedad de ninguna entidad u organización, y, por tanto, en las aplicaciones que corren en ella, tampoco hay ninguna entidad central que haga uso de los datos, o tenga un papel intermediario en cada una de las aplicaciones, como sí ocurre en las aplicaciones centralizadas.

Así pues, las partes involucradas en la aplicación son:

- **Usuarios:** Interactúan con la DApp. Serán quienes publiquen inmuebles y quienes realicen las acciones de comprar o alquilar.
- **Billetera:** Plugin que se conectará a la DApp, en forma de extensión de navegador, lo cual permitirá a los usuarios que conecten sus cuentas para poder crear un usuario en la plataforma y que este sea vinculado a la cuenta conectada, para poder realizar transacciones. En el punto de implementación, presentaremos Metamask [3], el plugin que utilizaremos como billetera.
- **Smart contracts:** Registran datos y eventos en la blockchain.
- **Almacenamiento distribuido:** Será necesario almacenar cierta información, y al no contar con un servidor centralizado que disponga de una base de datos tradicional, se hará uso de IPFS [4]. InterPlanetary File System es un protocolo y red P2P distribuida y direccionable por contenido, gracias a una función de hash. Permite almacenar y compartir información basándose en un sistema de archivos distribuidos.
- **Datos externos:** En ocasiones se requerirá consumir información del exterior de la cadena para realizar comprobaciones y modificaciones de datos, verificando así que cierta información es veraz.

2.2 ESPECIFICACIÓN

Desde la DApp, los usuarios tendrán la posibilidad de publicar propiedades de las cuales son dueños, una vez se autentiquen en ella, mediante la creación de una cuenta de usuario vinculada a una cuenta de usuario blockchain almacenada en una billetera. La publicación de una propiedad puede tener tres objetivos diferentes:

- **Vender una propiedad:** Un usuario podrá publicar una propiedad, con el objetivo de ponerla a la venta.
- **Alquilar una propiedad:** Un usuario podrá publicar una propiedad, con el objetivo de alquilarla hasta una fecha determinada.
- **Alquilar una propiedad de manera tokenizada:** La tokenización hace referencia a la emisión de tokens que representan activos reales, en este caso, un usuario podrá publicar una propiedad, con el objetivo de alquilarla de manera tokenizada, hasta una fecha determinada, o lo que es lo mismo, alquilar la propiedad en fracciones, o tokens, dónde el número de fracciones no podrá superar en ningún caso al número de habitaciones de la propiedad en cuestión.

Este último punto es el más innovador, y a su vez, el que más difiere en comparación a las plataformas inmobiliarias tradicionales. Al fin y al cabo, no deja de ser cómo un alquiler por habitaciones, dónde el usuario propietario, podrá tener la opción de fraccionar el alquiler de su inmueble en diferentes partes, y el usuario arrendatario, podrá escoger entre alquilar desde una habitación, hasta un máximo del número de habitaciones que tenga disponibles el inmueble.

2.3 ARQUITECTURA

La arquitectura que seguirá el proyecto será la siguiente:

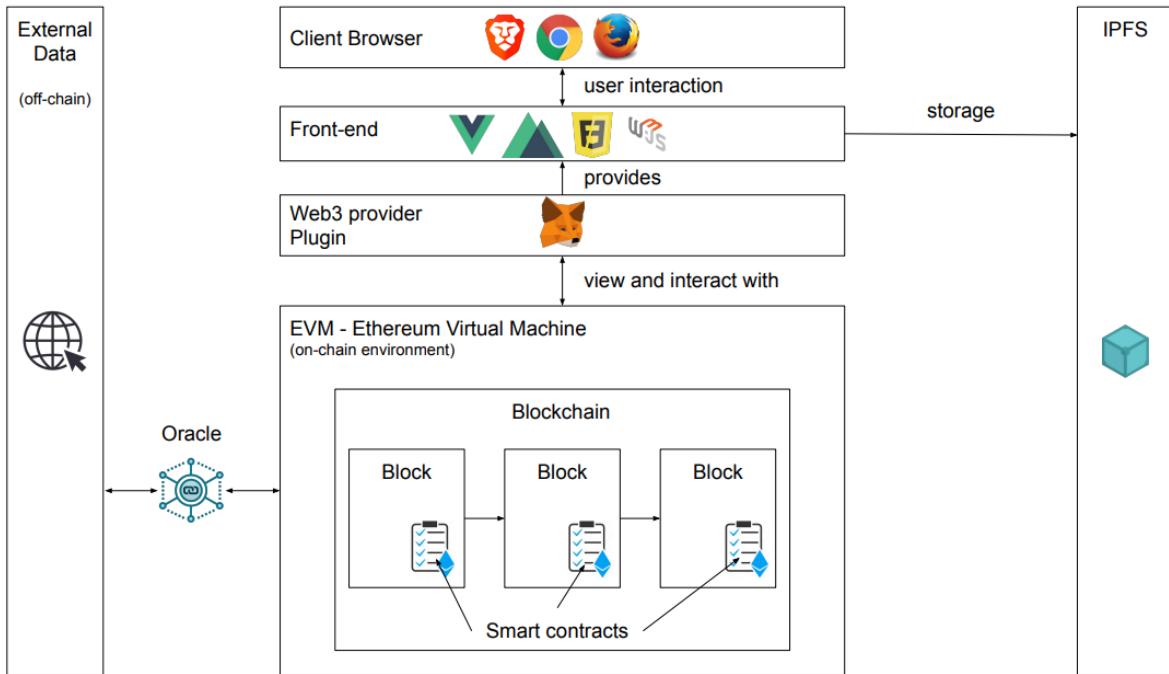


Figura 4: Arquitectura de la DApp
Fuente: Elaboración propia

Cómo podemos ver, en la “figura 4” los usuarios interactúan con la interfaz gráfica del lado cliente (Front-end) de la plataforma mediante un navegador web.

A su vez, el Front-end, se conectará con una extensión de navegador que permitirá conectar al cliente con la blockchain, mediante una cuenta de usuario que le permitirá realizar transacciones.

Como se ha comentado en el apartado “2.1”, en ciertos casos de uso será necesario almacenar información, y al no contar con un servidor centralizado detrás, dicha información se guardará de forma distribuida mediante el protocolo IPFS.

Finalmente, en ocasiones también será necesario consultar datos externos como APIs [5] que provean de cierta información a la red, para lo que se utilizarán oráculos que hagan de intermediarios entre el exterior (entorno off-chain) y la blockchain (entorno on-chain).

Tras esta introducción dónde se muestra la arquitectura completa de la DApp, y damos a conocer las partes involucradas, poniendo en contexto el papel que tiene cada una de ellas, en una visión general, vamos a pasar a comentar los casos de uso que existen dentro de la plataforma.

Posteriormente, volveremos a adentrarnos en cada una de las tecnologías anteriores, en esta ocasión, detallando de manera técnica cómo funciona cada una y como se integran ellas.

2.4 CASOS DE USO

Tras observar un plano general de la plataforma, en el presente apartado se especifican los casos de uso existentes seguidos de su diagrama de secuencia:

Caso de uso	Conecitar billetera (figura 5)
Precondición	<ul style="list-style-type: none">• El usuario debe tener creada una billetera para conectar una cuenta a la DApp
Eventos	<ol style="list-style-type: none">1. El usuario clica en el botón para conectar su billetera2. Se despliega una ventana para escoger la cuenta que se desea conectar3. El usuario escoge la cuenta que desea conectar4. La billetera escogida se conecta a la extensión de navegador5. La DApp guarda la clave pública de la cuenta conectada hasta que el usuario vincule otra distinta o desvincula la actual desde la extensión6. Se muestra un mensaje en pantalla indicando al usuario la clave pública de la cuenta que ha conectado a la aplicación
Posibles condicionales	<ol style="list-style-type: none">1.1. El usuario no tiene instalada la extensión de navegador necesaria para conectar su billetera a la plataforma<ol style="list-style-type: none">1.1.1. Se muestra un mensaje de error

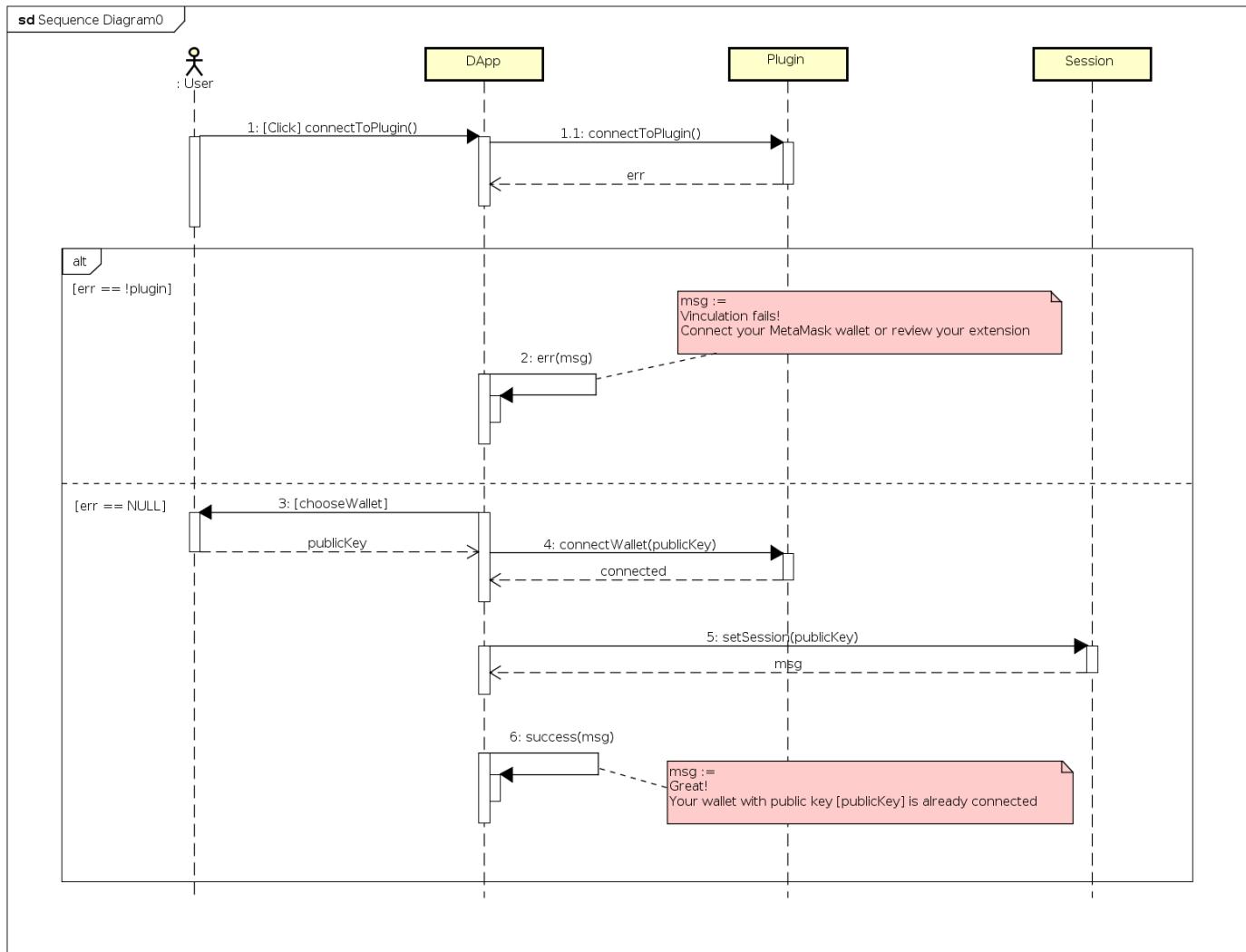


Figura 5: Diagrama de secuencia caso de uso “Conectar Billetera”
Fuente: Elaboración propia

Caso de uso	Registrar usuario (figura 6)
Precondición	<ul style="list-style-type: none">• Se debe tener conectada una cuenta para poder registrar un usuario
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre el botón que le envía a la pantalla de registro2. El usuario introduce los datos que la plataforma le requiere3. El usuario clica sobre el botón para completar el proceso de registro4. Se registra la información del nuevo usuario en un smart contract5. Se muestra un mensaje que indica que el usuario se ha creado correctamente
Posibles condicionales	<ol style="list-style-type: none">3.1. Billetera no conectada<ol style="list-style-type: none">3.1.1. Se muestra un mensaje de error4.1. Ya existe un usuario vinculado a la cuenta actual<ol style="list-style-type: none">4.1.1. Se muestra un mensaje de error

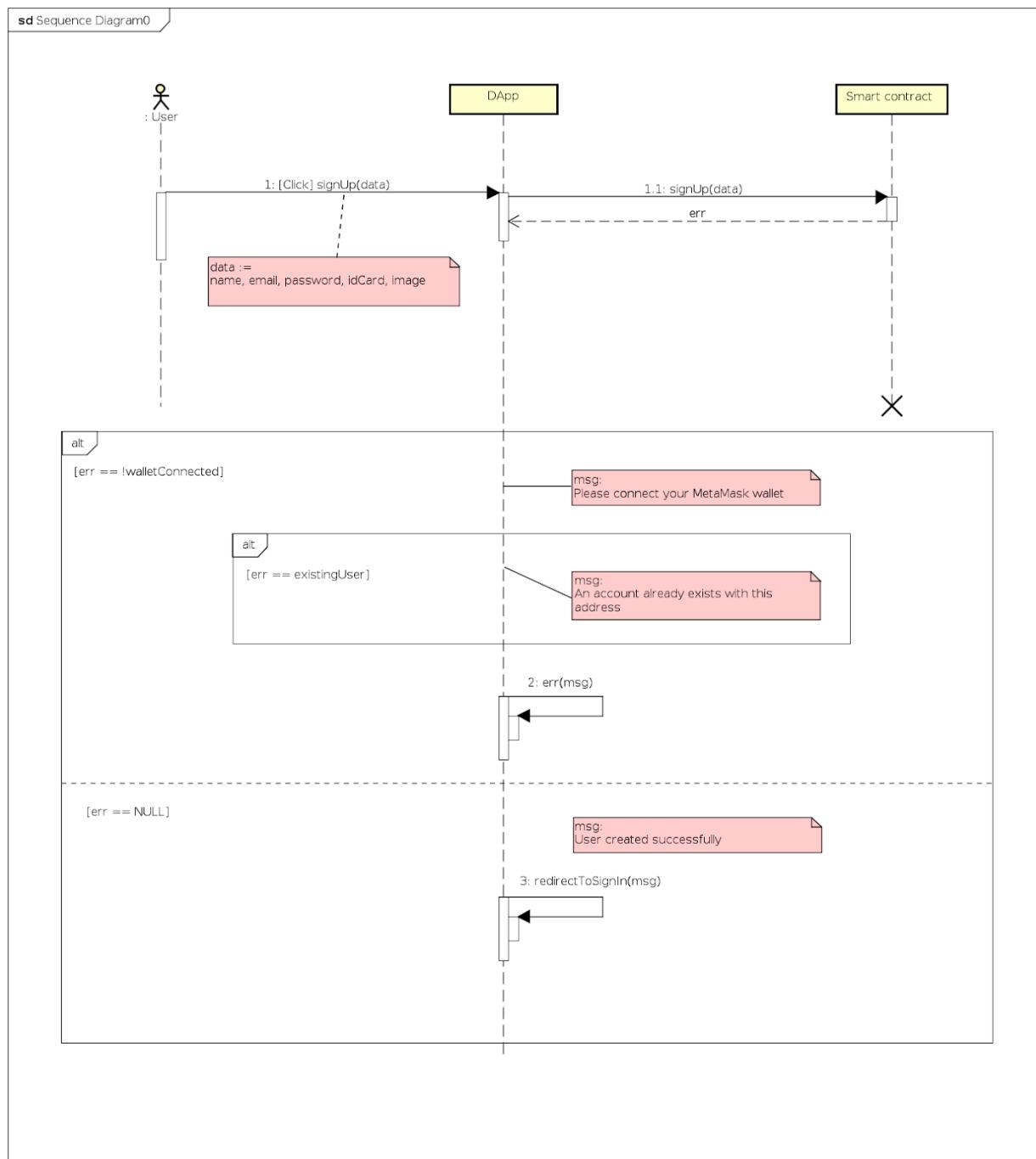


Figura 6: Diagrama de secuencia caso de uso "Registrar usuario"
Fuente: Elaboración propia

Caso de uso	Login usuario (figura 7)
Precondición	<ul style="list-style-type: none">• Se debe tener conectada una cuenta• El usuario debe estar registrado vinculado a una cuenta mediante un plugin
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre el botón que le envía a la pantalla de acceso2. El usuario introduce su contraseña3. Se registra el login del usuario en un smart contract4. La DApp guarda la sesión hasta que el usuario se desconecte5. Se redirige al usuario a la pantalla de inicio
Posibles condicionales	<ol style="list-style-type: none">2.1. El usuario introduce una contraseña incorrecta<ol style="list-style-type: none">2.1.1. Se muestra un mensaje de error

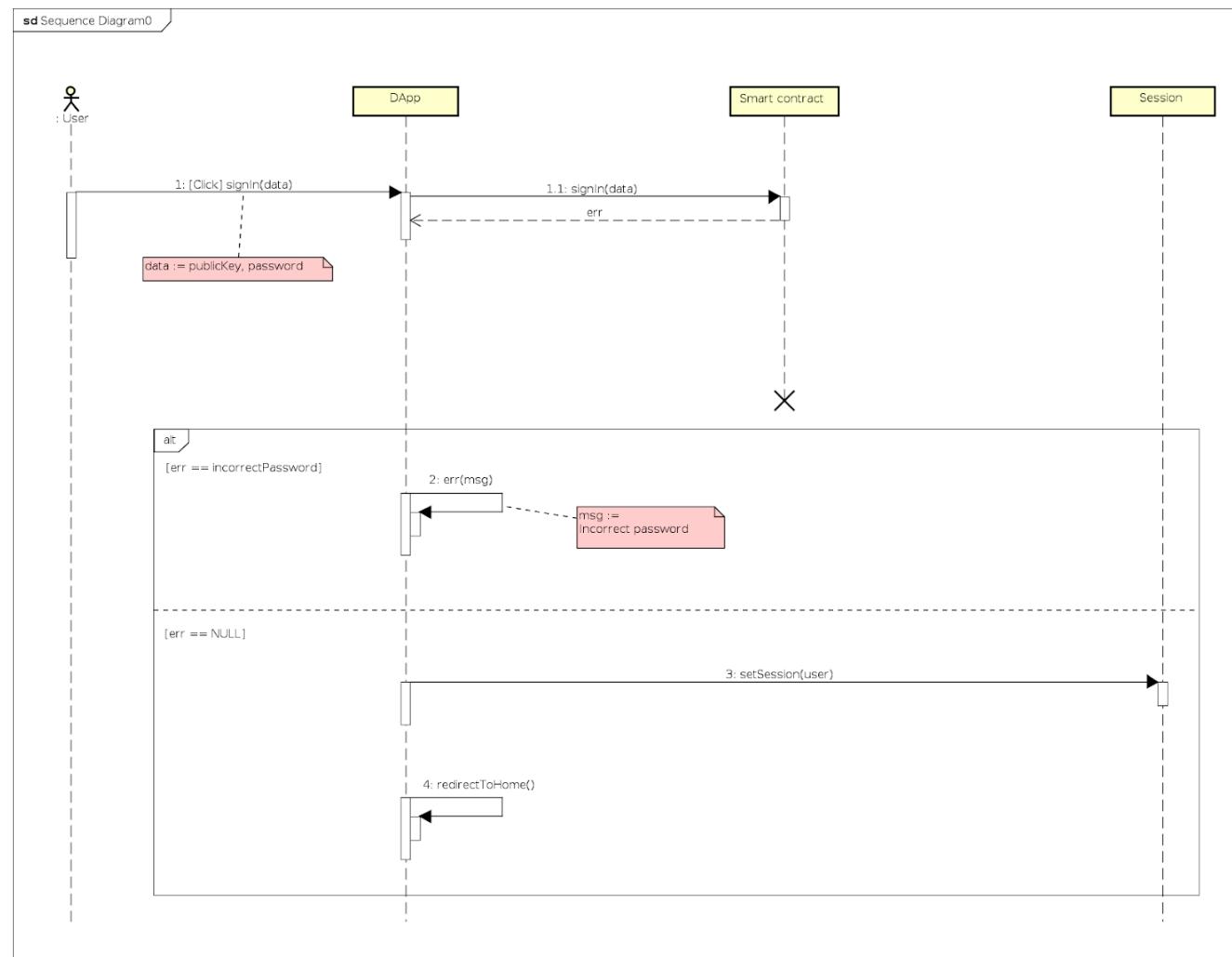


Figura 7: Diagrama de secuencia caso de uso "Login usuario"
Fuente: Elaboración propia

Caso de uso	Consultar perfil usuario (figura 8)
Precondición	<ul style="list-style-type: none">• El usuario debe estar logueado
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre su perfil de usuario en la pantalla de inicio2. Se muestran en pantalla todos sus datos a excepción de la contraseña
Posibles condicionales	-

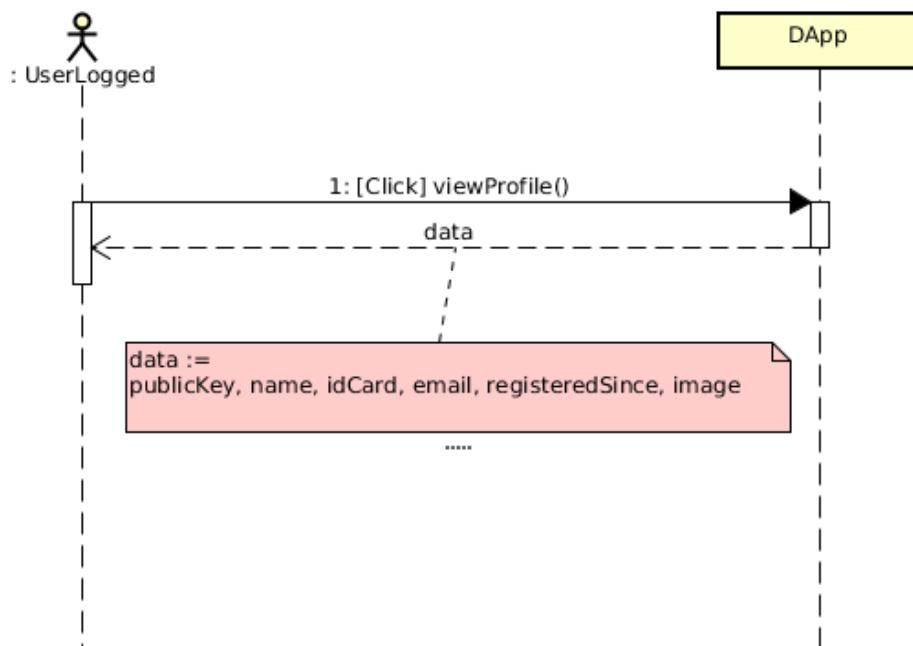


Figura 8: Diagrama de secuencia caso de uso "Consultar perfil usuario"
Fuente: Elaboración propia

Caso de uso	Logout usuario (figura 9)
Precondición	<ul style="list-style-type: none">• El usuario debe estar logueado
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre el botón de logout2. Se muestra un mensaje en pantalla para que el usuario verifique si realmente quiere cerrar la sesión3. El usuario verifica el cierre de sesión4. Se registra el cierre de sesión en un smart contract5. La DApp elimina la sesión actual de usuario6. Se redirige al usuario a la pantalla de inicio
Posibles condicionales	<ol style="list-style-type: none">2.1. El usuario finalmente decide no cerrar sesión<ul style="list-style-type: none">2.1.1. Se queda en la pantalla dónde estaba

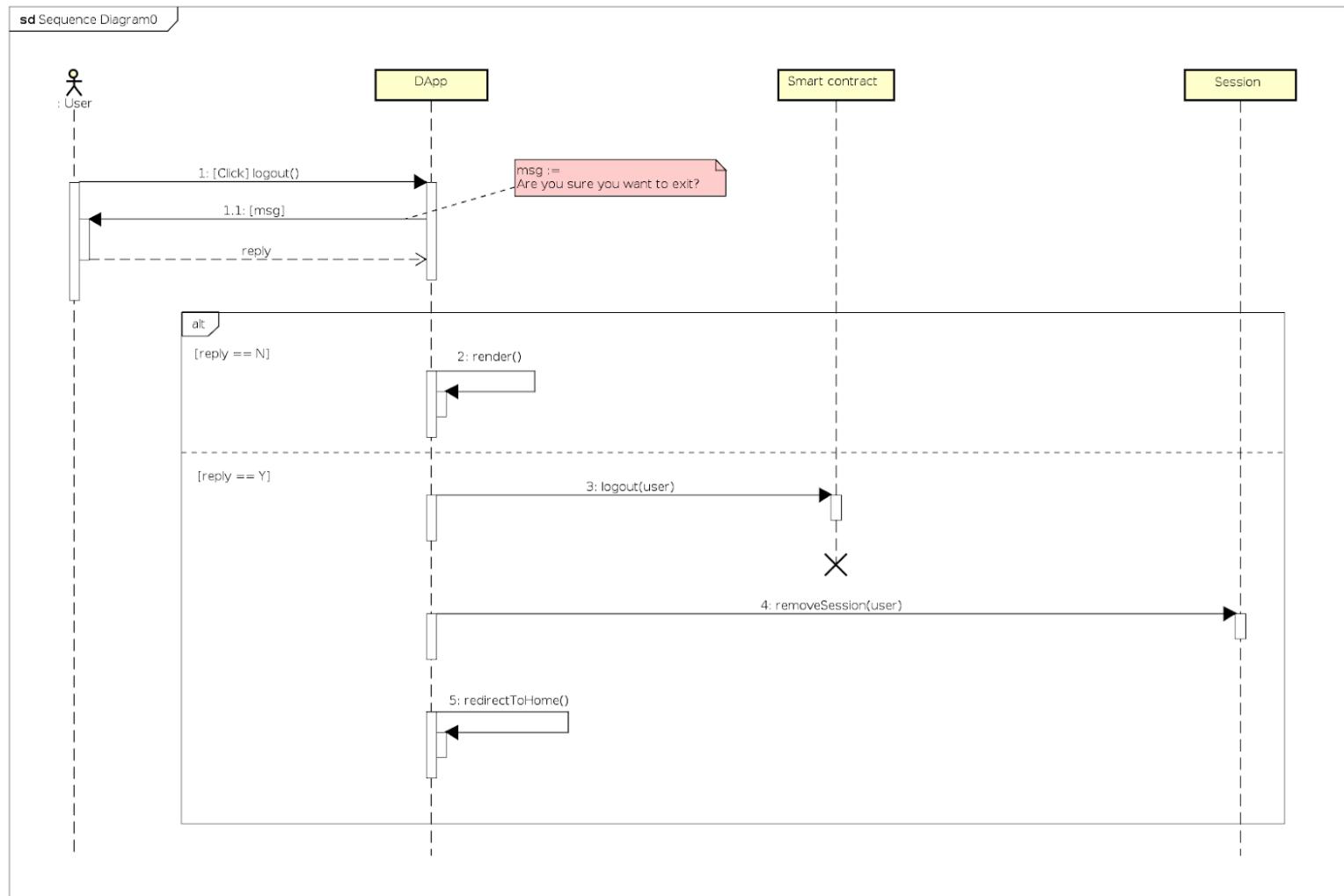


Figura 9: Diagrama de secuencia caso de uso "Logout usuario"
Fuente: Elaboración propia

Caso de uso	Publicar propiedad (figura 10)
Precondición	<ul style="list-style-type: none">• El usuario debe estar logueado
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre el botón de “publicar”2. El usuario introduce los datos del inmueble a publicar3. Se comprueba que el usuario logueado es realmente el propietario del inmueble consultando datos externos a la blockchain4. La imagen del inmueble subida por el usuario, se almacena de manera distribuida en la red5. Se registra la información del nuevo inmueble en un smart contract6. Se muestra un mensaje en pantalla y se redirige al usuario a la pantalla de “propiedades”
Posibles condicionales	<ol style="list-style-type: none">2.1. El inmueble a publicar ya está publicado en la plataforma<ol style="list-style-type: none">2.1.1. Se muestra un mensaje de error3.1. El usuario no es el propietario del inmueble que intenta publicar<ol style="list-style-type: none">3.1.1. Se muestra un mensaje de error

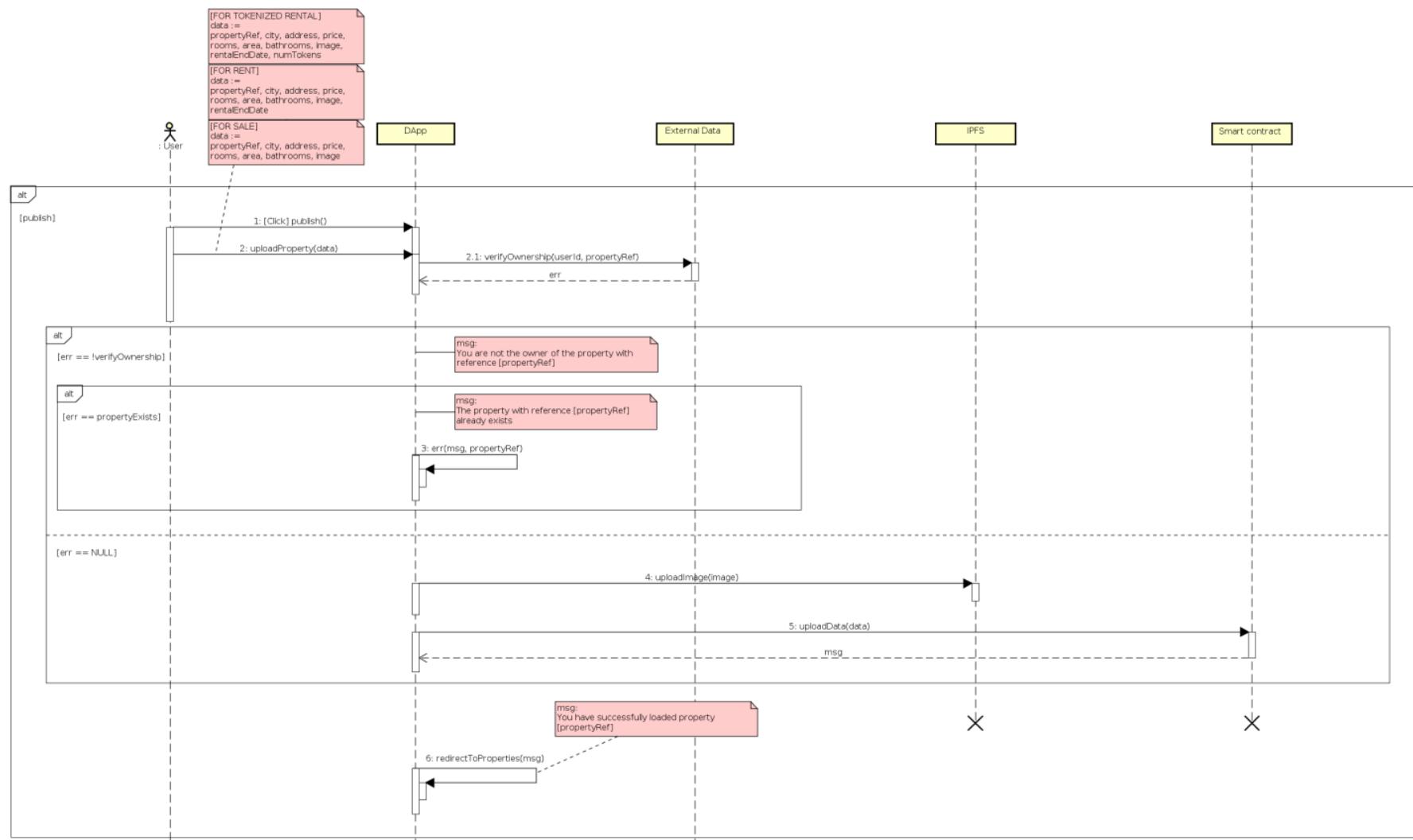


Figura 10: Diagrama de secuencia caso de uso "Publicar propiedad"
Fuente: Elaboración propia

Caso de uso	Comprar propiedad (figura 11)
Precondición	<ul style="list-style-type: none"> • El usuario debe estar logueado • El usuario no debe ser el propietario de la propiedad
Eventos	<ol style="list-style-type: none"> 1. El usuario clica sobre el botón de “propiedades” 2. El usuario pulsa el botón de “Comprar propiedad” 3. Se transfiere el valor del inmueble desde la billetera del usuario comprador hacia la billetera del usuario propietario 4. Se genera un contrato de compraventa con los datos del usuario comprador y el usuario propietario y se almacena de manera distribuida en la red 5. El propietario del inmueble se actualiza en un smart contract 6. Se actualiza el propietario del inmueble en datos externos a la blockchain 7. Se muestra un mensaje en pantalla indicando que la transacción ha finalizado correctamente 8. Se muestra un mensaje en pantalla dando la posibilidad de descargar el contrato de compraventa, además se le indica al usuario el hash del contrato almacenado de manera distribuida 9. Se refresca la pantalla
Posibles condicionales	<ol style="list-style-type: none"> 3.1. El usuario comprador no tiene suficientes fondos en su billetera <ol style="list-style-type: none"> 3.1.1. Se muestra un mensaje de error 8.1. El usuario decide descargar el contrato de compraventa <ol style="list-style-type: none"> 8.1.1. El contrato se descarga en el equipo del usuario 9.1. La propiedad aparece en estado “vendida”

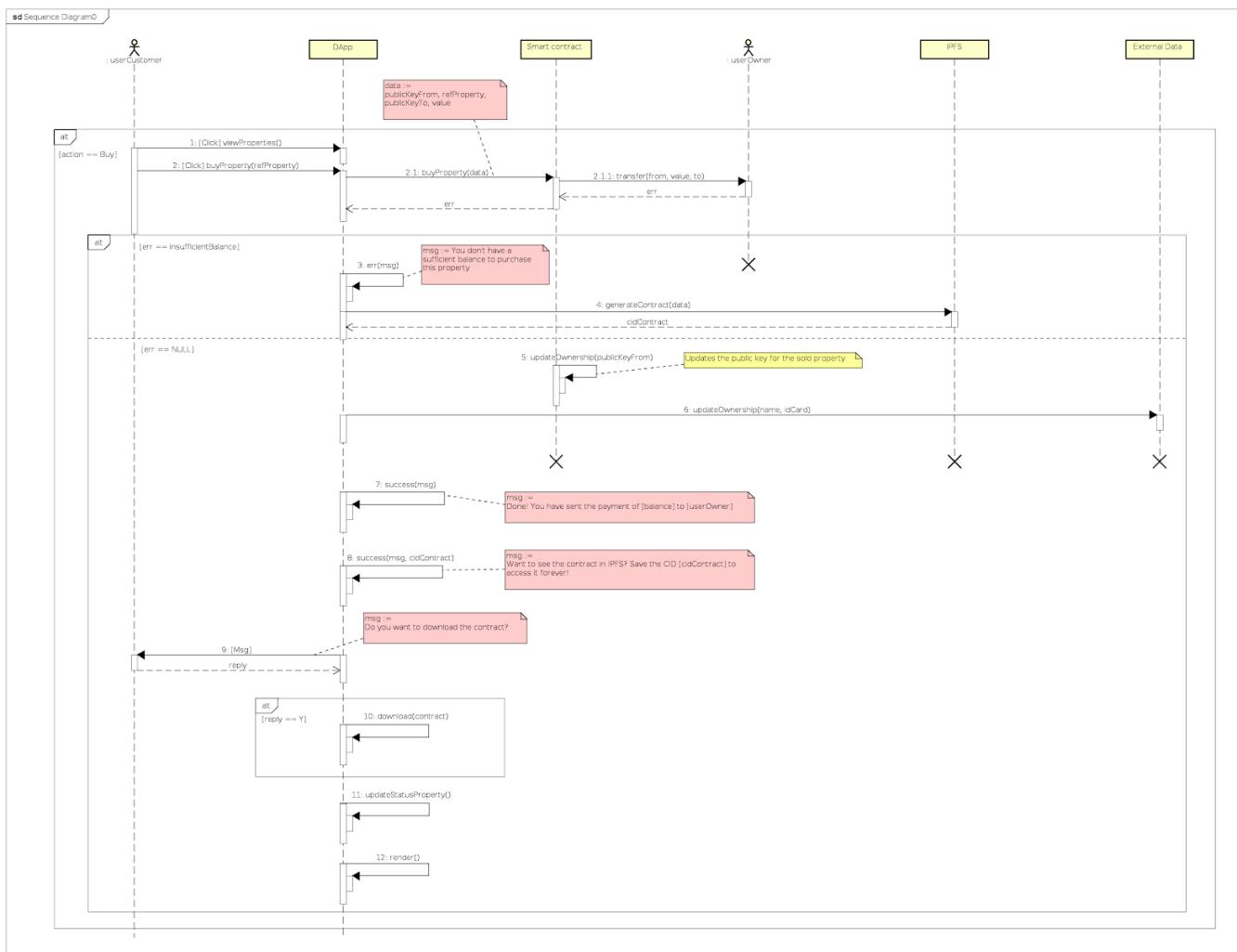


Figura 11: Diagrama de secuencia caso de uso "Comprar propiedad"
Fuente: Elaboración propia

Caso de uso	Alquilar propiedad (figura 12)
Precondición	<ul style="list-style-type: none"> • El usuario debe estar logueado • El usuario no debe ser el propietario de la propiedad
Eventos	<ol style="list-style-type: none"> 1. El usuario clica sobre el botón de “propiedades” 2. El usuario pulsa el botón de “Alquilar propiedad” 3. Se transfiere el valor de alquiler fijado por el propietario, desde la billetera del usuario inquilino hacia la billetera del usuario propietario 4. Se genera un contrato de alquiler con los datos del usuario inquilino y el usuario propietario y se almacena de manera distribuida en la red 5. Se muestra un mensaje en pantalla indicando que la transacción ha finalizado correctamente 6. Se muestra un mensaje en pantalla dando la posibilidad de descargar el contrato de alquiler, además se le indica al usuario el hash del contrato almacenado de manera distribuida 7. Se refresca la pantalla
Posibles condicionales	<ol style="list-style-type: none"> 3.1. El usuario comprador no tiene suficientes fondos en su billetera <ol style="list-style-type: none"> 3.1.1. Se muestra un mensaje de error 6.1. El usuario decide descargar el contrato de compraventa <ol style="list-style-type: none"> 6.1.1. El contrato se descarga en el equipo del usuario 7.1. La propiedad aparece en estado “alquilada”

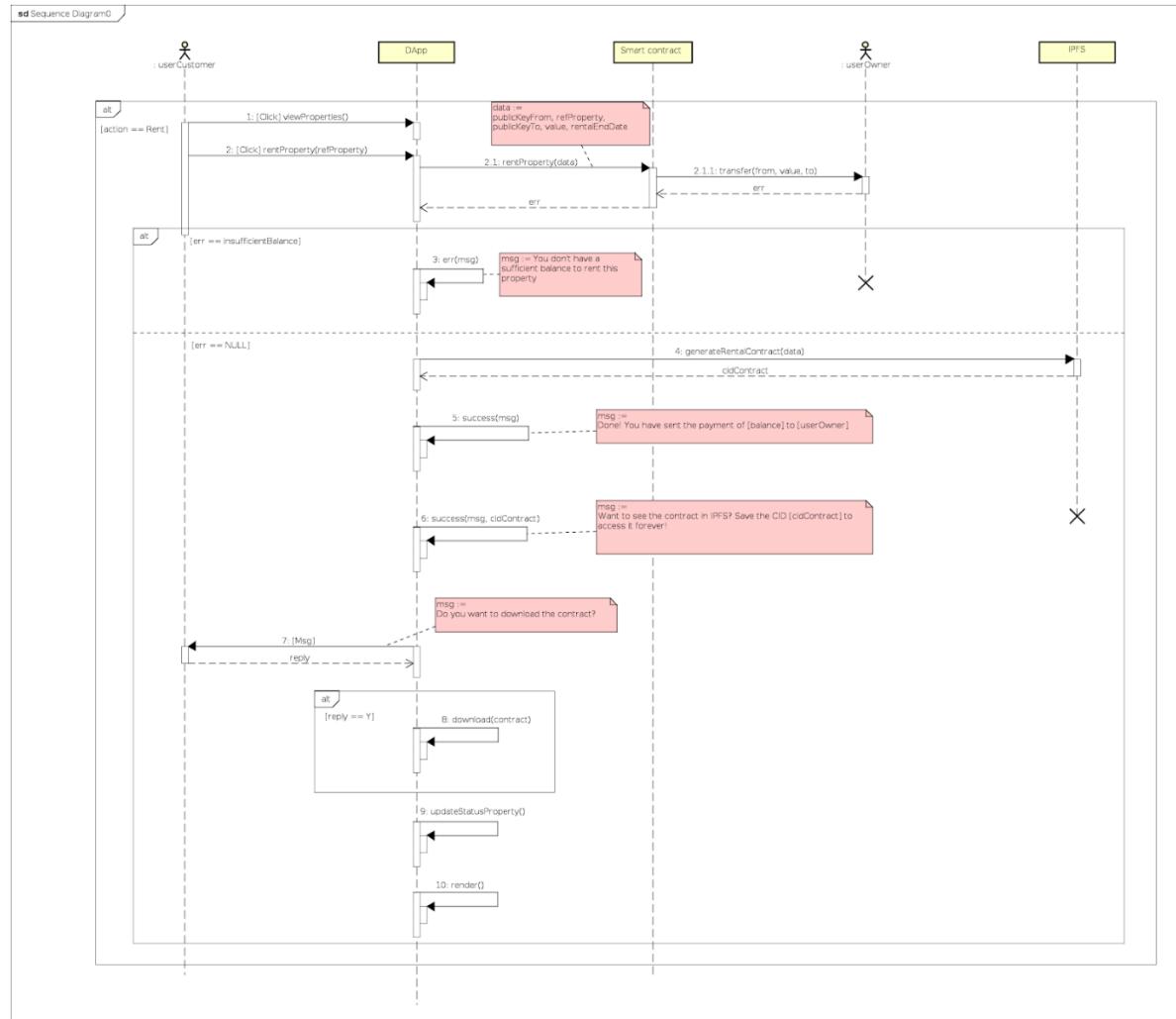


Figura 12: Diagrama de secuencia caso de uso "Alquilar propiedad"
Fuente: Elaboración propia

Caso de uso	Comprar tokens (figura 13)
Precondición	<ul style="list-style-type: none"> • El usuario debe estar logueado • El usuario no debe ser el propietario de la propiedad
Eventos	<ol style="list-style-type: none"> 1. El usuario clica sobre el botón de “propiedades” 2. El usuario escoge el número de tokens que desea adquirir 3. El usuario pulsa el botón de “Comprar tokens” 4. Se transfiere el valor del número de tokens escogidos, desde la billetera del usuario inquilino hacia la billetera del usuario propietario 5. Se genera un contrato de compra de tokens con los datos del usuario comprador y el usuario propietario y se almacena de manera distribuida en la red 6. Un smart contract genera un token no fungible (NFT) y lo envía a la billetera del usuario comprador 7. El número de tokens disponibles disminuye 8. Se muestra un mensaje en pantalla indicando que la transacción ha finalizado correctamente 9. Se muestra un mensaje en pantalla dando la posibilidad de descargar el contrato de compra de tokens, además se le indica al usuario el hash del contrato almacenado de manera distribuida 10. Se refresca la pantalla
Posibles condicionales	<ol style="list-style-type: none"> 4.1. El usuario comprador no tiene suficientes fondos en su billetera <ol style="list-style-type: none"> 4.1.1. Se muestra un mensaje de error 9.1. El usuario decide descargar el contrato de compraventa <ol style="list-style-type: none"> 9.1.1. El contrato se descarga en el equipo del usuario 10.1. El número de tokens disponibles es menor a 1 <ol style="list-style-type: none"> 10.1.1. La propiedad aparece en estado “alquilada”

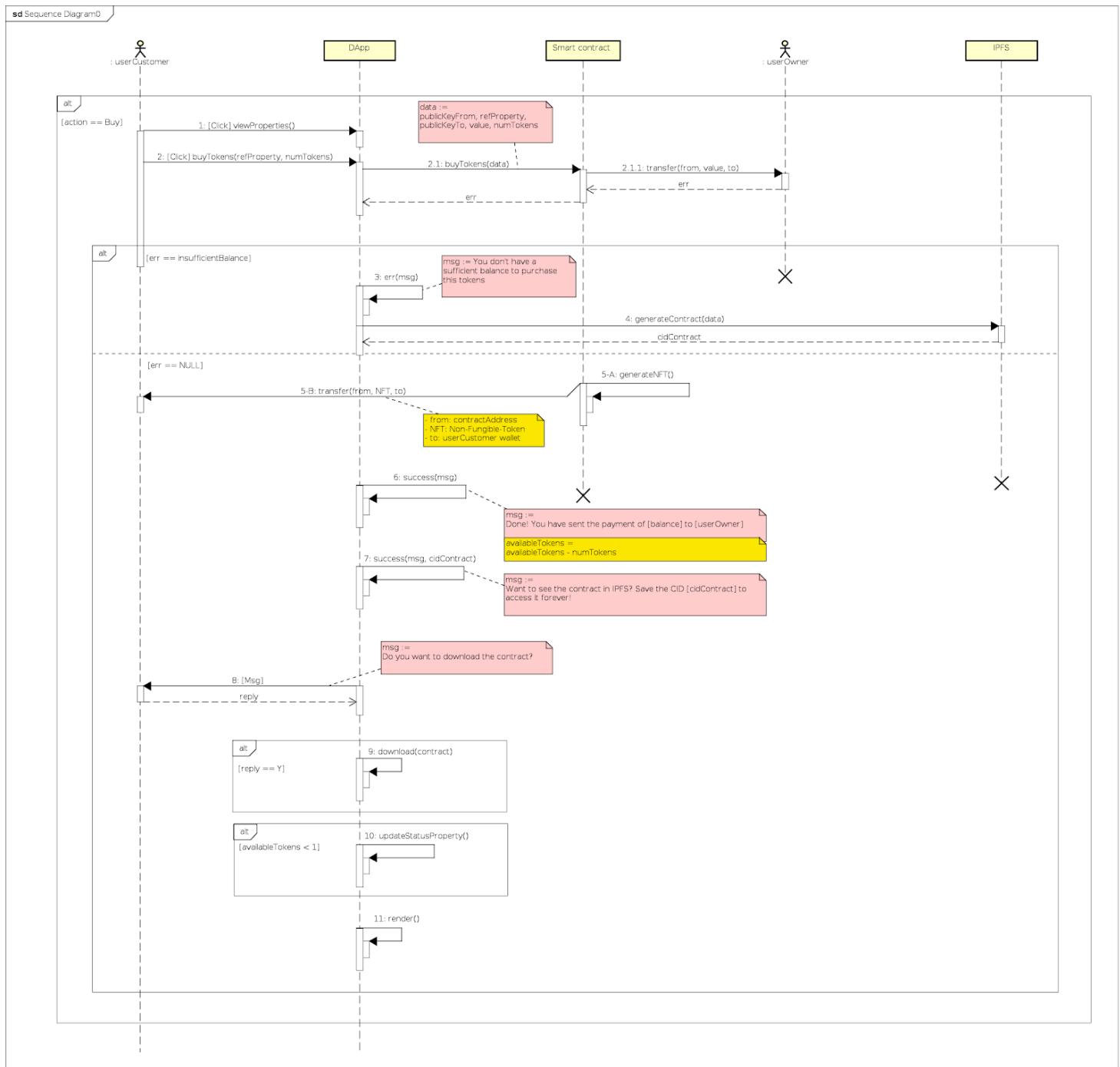


Figura 13: Diagrama de secuencia caso de uso "Comprar tokens"
Fuente: Elaboración propia

Caso de uso	Eliminar propiedad (figura 14)
Precondición	<ul style="list-style-type: none">• El usuario debe estar logueado• El usuario debe ser el propietario de la propiedad
Eventos	<ol style="list-style-type: none">1. El usuario clica sobre el botón de “propiedades”2. El usuario escoge la propiedad que desea eliminar de la plataforma3. Se muestra un mensaje en pantalla para que el usuario verifique si realmente quiere eliminar la propiedad4. El usuario acepta y se completa el proceso de eliminación5. Se registra la eliminación de la propiedad en un smart contract6. Se muestra un mensaje en pantalla indicando la referencia de la propiedad que ha sido eliminada7. Se refresca la pantalla
Posibles condicionales	<ol style="list-style-type: none">3.1. El usuario finalmente decide no eliminar la propiedad<ol style="list-style-type: none">3.1.1. Se queda en la pantalla “propiedades”

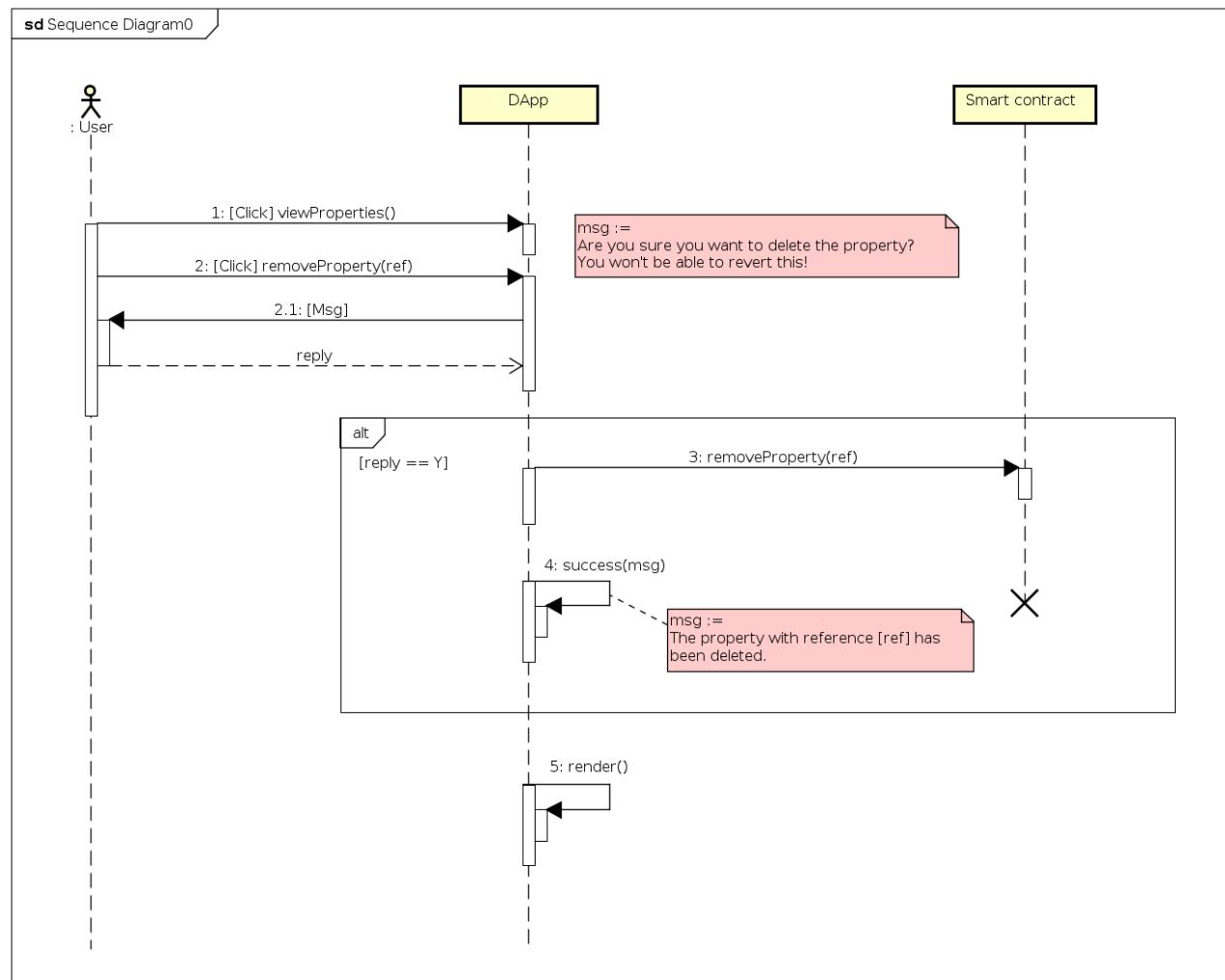


Figura 14: Diagrama de secuencia caso de uso "Eliminar propiedad"
Fuente: Elaboración propia

3. TECNOLOGÍAS DESCENTRALIZADAS

3.1 BLOCKCHAIN

Es la parte fundamental del proyecto, dónde correrá la DApp. Pese a que en el apartado “1.1.2”, ya se ha introducido este término, a continuación, vamos a ampliar la información sobre el mismo.

3.1.1 CARACTERÍSTICAS

- **Inmutabilidad:** Los contratos inteligentes y las transacciones realizadas, quedan almacenadas para siempre en la cadena de bloques.
- **Transparencia:** Las transacciones realizadas son públicas, cualquiera puede acceder a ellas y consultarlas.
- **Integridad:** Los nodos que participan en la red, poseen una réplica en todo momento del estado actual de la misma, y por tanto, validan la integridad de las transacciones y rechazan posibles transacciones malintencionadas.
- **Veracidad:** Únicamente participan dos partes en una transacción, sin intermediarios, por lo que ambas partes deciden si ejecutarla.
- **Anonimato:** Ofrece la posibilidad de realizar transacciones de forma anónima. Cada participante de la red se identifica mediante una clave pública, otros datos como el nombre completo y la dirección de correo electrónico se transforman en un hash criptográfico, el cuál es visible para todos participantes de la cadena de bloques.
- **Confiabilidad:** A diferencia de las bases de datos tradicionales, dónde se asume que todos los participantes son de confianza, la idea de blockchain consiste en ofrecer un protocolo de consenso entre los participantes de la red, el cual sirve para evitar que bloques con información no veraz sean añadidos a la cadena, o que, en caso de ser añadidos sean rechazados por el resto de nodos.

3.1.2 TIPOS DE BLOCKCHAIN

Existen diferentes tipos de cadenas de bloques:

- **Públicas:** Son las cadenas de bloques más habituales, y a las que nos solemos referir cuando nombramos a blockchain. Son aquellas a las que cualquier persona tiene acceso, ya sea como participante, mediante un nodo, o simplemente como usuario. Generalmente, el proceso para participar en una cadena de bloques pública, es descargar la aplicación correspondiente y conectarse.

El funcionamiento de las cadenas de bloques públicas es completamente transparente y abierto, sin entidades centralizadas, donde el mantenimiento económico de la red depende de si misma, gracias a qué cada red cuenta con su propio token, o “criptomoneda”, sobre el cual se realizan las transacciones y los pagos de comisiones de las mismas.

Ejemplo: Bitcoin [6]

- **Privadas o permisionadas:** Mucho menos extendidas, nacen tras la expansión inicial de la tecnología blockchain, donde muchas empresas vieron una oportunidad de negocio. En este tipo de cadenas de bloques, el control está reducido a una única entidad que se encarga de mantener la cadena, dar permiso a los usuarios que quieran participar o proponer transacciones.

Ejemplo: Hyperledger [7]

- **Federadas o híbridas:** Son una fusión entre los dos tipos anteriores. En este tipo de cadenas de bloques, la participación en la red es privada, el control de los recursos de la red está administrado por una o varias organizaciones, que mantienen copias sincronizadas de la cadena y autorizan a los miembros que pueden participar en ellas.

Ejemplo: Alastria [8]

En el caso particular de este proyecto, nos vamos a centrar en las cadenas de bloques públicas, para no depender de ninguna entidad central y seguir los principios de la descentralización. De esta manera, dentro de las cadenas de bloques públicas, actualmente existen tres generaciones, cada una de ellas con características específicas:

- **Primera generación:** Establece por primera vez en la historia, una manera de realizar transacciones con dinero digital descentralizado, sin necesidad de terceros de confianza, creando un protocolo de consenso llamado Proof-of-Work (PoW).

Mediante este protocolo, cuando un usuario inicia una transacción en la red, existen participantes de la red que actúan como validadores, los cuales también se conocen como “mineros”, debido a que se tratan de supercomputadoras que intentan resolver cálculos matemáticos con el objetivo de encontrar un hash que comience con un número concreto de ceros, mediante la entrada de datos de todas las transacciones encapsuladas en el bloque, sumado a un número aleatorio, lo cual les permite agregar bloques a la red. Como recompensa por su trabajo, estos obtienen una recompensa de cada transacción y un porcentaje de tokens de la red en cuestión. Sin embargo, tiene algunos inconvenientes como el gran gasto de energía que supone tener supercomputadoras encendidas resolviendo algoritmos durante períodos prolongados.

La cadena de bloques de primera generación que impulsó la tecnología blockchain fue Bitcoin, la cual empezó a funcionar en 2009, por lo que, como se puede ver, es una tecnología muy joven.

- **Segunda generación:** La gran diferencia respecto a las cadenas de bloques de primera generación, es que introducen los contratos inteligentes, gracias a ser cadenas que cuentan con un sistema de Turing complete [9], o lo que es lo mismo, sistemas que cuentan con un poder computacional equivalente a la máquina de Turing universal, lo cual les permite poder llegar a programar cualquier tipo de operación.

En resumidas cuentas, las cadenas de bloques de segunda generación pueden implementar estructuras de datos como, por ejemplo, los bucles, lo que les permite crear aplicaciones descentralizadas (DApps) o la tokenización digital de activos físicos. El avance y masificación de las cadenas de bloques, generaron dificultades en términos de escalabilidad, sostenibilidad y velocidad de transacciones.

Ethereum [10], la cual nació en 2013, es la base de la tecnología blockchain de segunda generación, siendo la primera cadena de bloques Turing complete.

- **Tercera generación:** Mantienen las bases de las cadenas de bloques de segunda generación, mientras buscan resolver problemas de escalabilidad (aumento del tamaño de bloques), sostenibilidad (reducción del consumo de energía), velocidad de transacciones (confirmaciones de red más rápidas y eficientes), interoperabilidad (creación de cadenas de cadenas de bloques), etc.

Como punto a destacar, introducen un nuevo protocolo de consenso llamado Proof-of-Stake (PoS), el cual fue diseñado para paliar los inconvenientes del protocolo PoW, en especial su gran consumo energético. En este caso, los participantes no actúan como mineros, sino que apuestan una cantidad que poseen del token nativo de la red, para optar a ser participantes, es decir, se incentiva a aquellos poseedores del token de la cadena.

Un ejemplo de blockchain de tercera generación es Cardano [11].

Tras hacer una distinción entre los diferentes tipos de cadenas de bloques y estudiar sus características, la escogida para desarrollar nuestro proyecto es Ethereum, debido a qué las cadenas de bloques de primera generación no permiten el despliegue de aplicaciones en su interior, y que las de tercera generación actualmente están en pleno desarrollo, no contando todavía con herramientas que faciliten las tareas de testing, implementación y despliegue a los desarrolladores.

3.1.3 ETHEREUM

El presente proyecto se ha decidido implementarlo en la red de Ethereum, por diferentes motivos:

- **Descentralización:** No hay ninguna entidad central detrás, al tratarse de una blockchain pública.
- **Smart contracts:** Permite el desarrollo de aplicaciones para que corran en su red, pudiendo implementar contratos inteligentes mediante un lenguaje de programación específico para ello.
- **Uso de Ether:** Posibilidad de realizar transacciones P2P mediante la criptodivisa Ether, la cual hace referencia al token nativo ETH, el cual es utilizado como medio de pago dentro de la red.

3.1.3.1 BLOQUES

Cada cadena de bloques tiene sus propias características, como hemos visto en puntos anteriores, por lo que, el contenido de los bloques también varía en función de la blockchain. En el caso de Ethereum, utiliza el algoritmo criptográfico Keccak-256, el cual derivado de la entrada de datos de cada bloque, devuelve una salida de 256 bits.

En el caso de Ethereum, la blockchain que vamos a utilizar, cada bloque contiene los siguientes campos:

- **Listado de transacciones:** Número de transacciones incluidas en el bloque.
- **Número de bloque:** Representa la longitud de bloques de la blockchain en cada momento.
- **Nonce:** Representa el número entero que los mineros deben encontrar y que, combinado con el resto de datos del bloque, permitan encontrar un hash valido para el bloque.
- **Dificultad de bloque:** Define cuán complejo es el algoritmo matemático que los mineros deben resolver para encontrar el Nonce objetivo de un bloque, y así poder minarlo.
- **State Root:** Es el estado completo de la red, incluye los saldos de cuentas, el almacenamiento de contratos, el código de los mismos y los Nonces de las cuentas.
- **Hash:** Identificador único del bloque, se genera mediante una entrada de datos que contiene el resto de datos del bloque.
- **Parent hash:** Identificador único del bloque predecesor, realiza la función de cadena y así es como los bloques quedan enlazados entre sí.

3.1.3.2 TIPOS DE CUENTA

Cómo hemos visto anteriormente, al tratarse de una blockchain de segunda generación, es capaz de correr aplicaciones descentralizadas en su interior y no simplemente realizar transacciones de criptomonedas. Pese a ello, la red cuenta con su propia criptomoneda, el Ether (símbolo ETH), la cual asegura el sistema financiero de la cadena de bloques de Ethereum.

Las transacciones de los tokens ETH se realizan entre cuentas blockchain, las cuales viven dentro de las billeteras. Existen dos tipos de cuentas:

- **Cuentas de usuario:** Está formada por un par criptográfico de claves: clave pública y clave privada. Estas claves permiten probar que una transacción está firmada por el remitente y previene falsificaciones, siendo la clave privada la utilizada para firmar transacciones, garantizando la custodia de los fondos de la cuenta.

La creación de una cuenta se puede realizar mediante ciertas librerías, las cuales, mediante una entrada de datos, la cual se suele denominar mnemónico o “frase semilla”, generan una clave privada que consta de 64 caracteres hexadecimales. Posteriormente, la clave pública se genera a partir de la clave privada, mediante un algoritmo de firma digital de curva elíptica (ECC), el cual se basa en funciones matemáticas simples de calcular en una dirección y muy difíciles de revertir. Del resultado obtenido, se obtiene una dirección de clave pública mediante los últimos 20 bytes de la clave pública y añadiendo 0x al inicio.

```
1 > personal.newAccount()  
2 Frase de contraseña:  
3 Repetir la frase de contraseña:  
4 "0x5e97870f263700f46aa00d967821199b9bc5a120"  
5  
6 > personal.newAccount("h4ck3r")  
7 "0x3d80b31a78c30fc628f20b2c89d7ddbf6e53cedc"  
8
```

Figura 15: Creación de una cuenta mediante la librería personal_newAccount

Fuente: ethereum.org

- **Cuentas de contrato:** Las cuentas donde el propietario es un contrato inteligente, por el contrario, poseen una dirección hexadecimal de 42 caracteres, la cual es asignada cuando un contrato inteligente se despliega en la cadena de bloques de Ethereum. La dirección se obtiene de la dirección de clave pública del creador del contrato y del número de transacciones enviadas desde esa dirección.

3.1.3.3 TOKEN ETH Y SUS ESTÁNDARES

Entre cuentas se realizan transacciones del token nativo ETH, el cual es un estándar **ERC-20**, el cual ofrece una nomenclatura fija y permite utilizar tokens con software de terceros, lo cual facilita la implementación a los desarrolladores de aplicaciones al utilizar dicho estándar. Sin embargo y pese a qué es muy útil, dista de ser perfecto, ya que cuenta con algunos fallos como la imposibilidad de transferir tokens a una dirección de clave pública perteneciente a un contrato inteligente.

Cabe comentar, que en Ethereum existen otros estándares de tokens además del ERC-20, los comentaremos a continuación:

- **ERC-223:** Solventa el error de diseño de ERC-20, permitiendo a los usuarios transferir tokens a billeteras pertenecientes a contratos inteligentes.
- **ERC-721:** Representa lo que se denominan tokens no fungibles (**Non-Fungible Token o NFT**), es decir, un token que puede poseer un valor distinto a otro token dentro de la misma aplicación/ecosistema.
Su objetivo es el de desarrollar tokens únicos dónde su valor intrínseco venga dado de su rareza. Su mayor utilidad reside en que permiten la tokenización de activos individuales únicos, lo cual inevitablemente, va avanzando hacia la tokenización de activos del mundo real. A diferencia de los estándares ERC-20 o ERC-223, no son divisibles o fraccionables, ni se deterioran o destruyen.

Cada vez que un usuario utiliza una aplicación de Ethereum o realiza un envío de ETH, paga una comisión de ETH, la cuál es un incentivo para el minero que procese y verifique el bloque que contenga dicha transacción.

El Ether, por su parte, es divisible, hasta 10^{18} veces, siendo el Wei la unidad más pequeña del Ether [12], y la unidad en la cual se representarán en la mayoría ocasiones el pago de comisiones en las transacciones.

3.1.3.4 ETHEREUM VIRTUAL MACHINE (EVM)

Como hemos comentado en apartados anteriores, las cadenas de bloques de segunda generación cuentan con un sistema Turing completo, específicamente en Ethereum hacemos referencia a la EVM. Las transacciones entre cuentas y el desarrollo de aplicaciones descentralizadas, es posible gracias a la máquina virtual de Ethereum, la cual funciona como una computadora descentralizada, sustentada por todos los nodos conectados a la red que ejecutan un cliente de Ethereum (cuenta de Ethereum, despliegue de contrato inteligente...).

Su propósito es mantener un funcionamiento continuo, ininterrumpido e inmutable, mediante un estado global que está sincronizado con el bloque actual de la cadena mediante el campo State Root visto en los puntos anteriores, por lo cual, es común en todos los nodos participantes.

La EVM convierte a Ethereum en algo más que una base de datos distribuida, asemejándose más a una máquina de estado distribuida. El estado de la red es una estructura de datos de pila (Stack) con un tamaño máximo de 1024 elementos y 256 bits, la cual contiene todas las cuentas y saldos, además del estado global de la máquina.

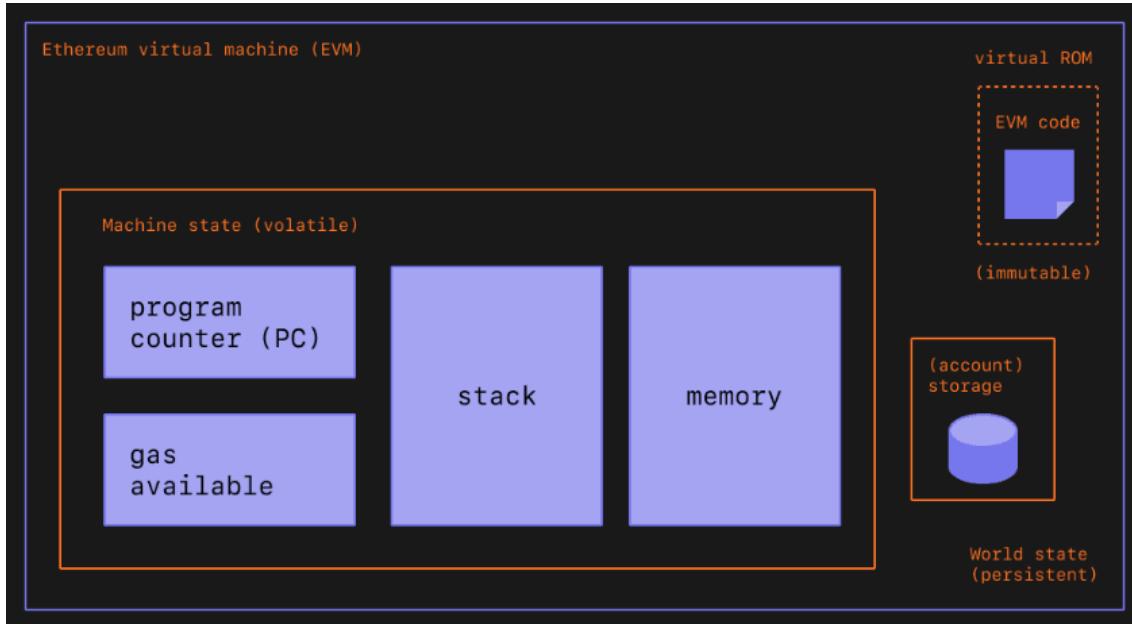


Figura 16: Arquitectura de la EVM

Fuente: ethereum.org

Cuando un usuario ejecuta transacciones, como por ejemplo, el despliegue de contratos inteligentes en la EVM, se hace mediante un lenguaje de programación de alto nivel creado para dicho propósito, Solidity [13], cuyo código se transforma en códigos de operación, como por ejemplo POP, PUSHn, ADD, AND, XOR, y así hasta 256 distintos, además de brindar a los usuarios la posibilidad de trabajar con direcciones de clave pública mediante el tipo de dato “address”, el cual contiene un valor de 20 bytes, el tamaño que ocupa una dirección de Ethereum.

Y una vez está el código transformado en Opcodes, este es compilador y se obtiene el llamado Bytecode, es decir, código de bajo nivel ilegible para humanos que la EVM ejecuta para realizar las operaciones específicas que el usuario ha programado inicialmente en Solidity.

```
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe608060405  
2348015600f57600080fd5b506004361060285760003560e01c8063037a417c14602d  
575b600080fd5b60336049565b6040518082815260200191505060405180910390f35  
b6000600190509056fea265627a7a7230582050d33093e20eb388eec760ca84ba30ec  
42dadbddeb8edf5cd8b261e89b8d4279264736f6c634300050a0032
```

Figura 17: Ejemplo de bytecode

Fuente: medium.com

También hay que añadir, que Ethereum apuesta por la transparencia y brinda a la EVM obtener el ABI (Application Binary Interface) a partir de un Bytecode descompilado, brindando la posibilidad de mantener de forma clara y abierta el contenido de un contrato inteligente.

```
1  [
2    {
3      "constant":true,
4      "inputs":[
5        ],
6      "name":"testFunc",
7      "outputs":[
8        {
9          "name":"",
10         "type":"int256"
11       }
12     ],
13     "payable":false,
14     "stateMutability":"pure",
15     "type":"function"
16   }
17 ]
```

Figura 18: Ejemplo de ABI derivado del bytecode de la “figura 17”
Fuente: medium.com

3.1.3.5 GAS

Llegados a este punto, vamos a introducir uno de los últimos conceptos de la EVM, el concepto de gas.

El gas hace referencia a la unidad que mide el esfuerzo computacional requerido para ejecutar operaciones específicas en la red, lo que directamente está asociado a una comisión en Ether, pese a qué habitualmente dichas comisiones de gas están indicadas en GWei (10-9 ETH). Las comisiones de gas, son necesarias para que las transacciones se lleven a cabo con éxito y para mantener la red segura, evitando spam o generación de desperdicio computacional mediante código (bucles infinitos o accidentales...).

Cada transacción contiene un campo llamado límite de gas, en el cual el usuario indica el máximo gas que quiere usar en dicha transacción. Si dicho gas no se utiliza, es devuelto al usuario al finalizar la transacción, por el contrario, si el límite de gas es menor al necesario para completar la transacción, esta se revierte, deshaciendo así todos los cambios en la cadena, pero sin devolver la comisión al usuario, puesto que el gas se habrá consumido durante el proceso computacional [14].

3.2 IPFS

A diferencia de blockchain, tecnología que ya habíamos presentado ampliamente en el apartado introductorio para poner en contexto el proyecto, vamos a presentar la segunda tecnología descentralizada utilizada, el protocolo InterPlanetary File System (IPFS). El propósito de IPFS es similar al de blockchain, crear una red de computadoras, o nodos, de alcance global, en este caso con el objetivo de permitir el almacenamiento de información de manera totalmente descentralizada, consiguiendo como resultado una red P2P global.

Actualmente cuando los usuarios navegan por Internet, utilizan el protocolo HTTP, el cual permite acceder a diferentes sitios webs mediante nombres de dominio que traducen direcciones IP, sin embargo, estas direcciones y dominios están administrados por servidores centralizados, lo cual está expuesto a la censura de las entidades o gobiernos que proveen dichos servicios.

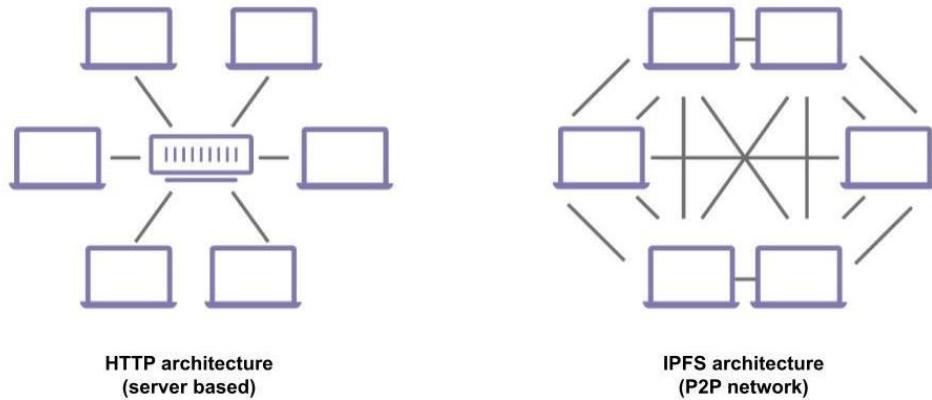


Figura 19: Comparación entre arquitecturas HTTP e IPFS
Fuente: Elaboración propia

Por el contrario, IPFS permite crear una red permanente y distribuida, utilizando un sistema de búsqueda por contenido. Esto quiere decir que, mientras que HTTP refiere y transmite dominios o datos situados en un servidor concreto, IPFS funciona únicamente haciendo referencia a un código hash (CID) que identifica a cada archivo compartido en la red.

En ese momento, el sistema envía el CID a la red y todos los nodos de la red empiezan a dar respuesta, y finalmente se crea una comunicación con el nodo más cercano que posea el CID introducido, debido a que el almacenamiento se realiza de manera distribuida, guardando ficheros divididos en fragmentos en diferentes nodos.

Cuando un nodo busca un archivo referenciado, realiza una consulta a los nodos cercanos para ver quién almacena dicho contenido. Una vez es encontrado, se guarda una copia de dicho archivo en caché, y todo nodo que ha buscado dicho archivo, se convierte en proveedor del mismo hasta que se borra su caché.

Para participar en la red mediante un nodo, los usuarios deben descargar el cliente IPFS desde su sitio oficial. Una vez descargado, IPFS provee a los usuarios de un entorno gráfico para ver el estado de su nodo, además de la posibilidad de utilizar línea de comandos.

A continuación, vamos mostrar un ejemplo sencillo, dónde un usuario cargará un fichero de muestra, el cual se almacenará de manera distribuida y posteriormente se accederá a él.

Inicialmente, para utilizar la línea de comandos que provee IPFS, se deberá iniciar un nodo local mediante el comando **ipfs daemon**.

Posteriormente se cargará el fichero, mediante el comando **ipfs add <filename>**, y se devolverá el hash CID que identifica al fichero, gracias al cual mediante el comando **ipfs cat <CID>**, se podrá obtener el contenido del fichero:

```
hector@Ubuntu-21:~$ cat saludoDesdeIpfs.txt
¡Este es un saludo desde IPFS para los lectores de la memoria del TFG de Héctor Montesinos! 🚀
hector@Ubuntu-21:~$ ipfs add saludoDesdeIpfs.txt
added QmNfu2a2puGUHY6D1fxhdKVfTC6SKoLYTfU5dREa7ERnjk saludoDesdeIpfs.txt
  99 B / 99 B [=====] 100.00%
hector@Ubuntu-21:~$ ipfs cat QmNfu2a2puGUHY6D1fxhdKVfTC6SKoLYTfU5dREa7ERnjk
¡Este es un saludo desde IPFS para los lectores de la memoria del TFG de Héctor Montesinos! 🚀
hector@Ubuntu-21:~$
```

Figura 20: Subida de un fichero a IPFS y posterior visualización

Fuente: Elaboración propia

El usuario también podrá visualizar el fichero desde el entorno gráfico que proporciona IPFS, dónde el nodo local, por defecto, corre en el puerto 5001. El usuario tiene la posibilidad de introducir el hash CID en un buscador y además puede visualizar el fichero cargado en el apartado “archivos”:



Figura 21: Visualización de fichero subido a IPFS, filtrado mediante su hash CID
Fuente: Elaboración propia

Por último, el fichero también será visible desde un enlace web, introduciendo el hash CID en la URL <https://ipfs.io/ipfs/<hash>>:



Figura 22: Visualización de fichero subido a IPFS mediante URL
Fuente: Elaboración propia

Cabe destacar que, IPFS posee un sistema de almacenamiento en el cual, los ficheros a los cuales no se ha accedido regularmente son borrados para evitar un almacenamiento innecesario. Para ello, existe la funcionalidad “pinning”, la cual permite fijar ficheros y almacenarlos de manera persistente utilizando su propio espacio de nodo.

4. IMPLEMENTACIÓN

Tras tener una vista general de las tecnologías descentralizadas y adentrarnos en ellas para conocerlas, pasamos al apartado de implementación, dónde se explicará el desarrollo del proyecto y las herramientas que han sido utilizadas para cada propósito. El diagrama general de la arquitectura, con las tecnologías utilizadas es el siguiente:

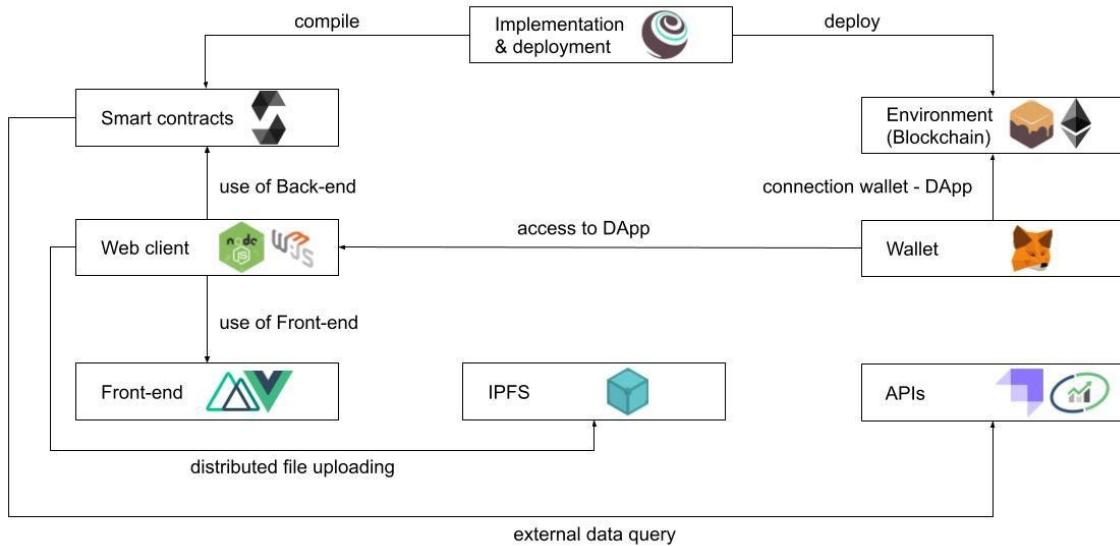


Figura 23: Arquitectura según las tecnologías utilizadas
Fuente: Elaboración propia

Para explicar el desarrollo de forma ordenada, podemos desplegar el diagrama anterior en tres partes principales:

- **Blockchain**: Toda la implementación referente a la red donde se desplegará la DApp. Engloba los contratos inteligentes, la implementación y despliegue y la configuración del entorno de trabajo.
- **Cliente web**: Entorno mediante el cual los usuarios interactúan con la plataforma. Engloba la creación del cliente web y su interfaz de usuario (Front-end).
- **Intermediarios**: Herramientas y aplicaciones que permiten interconectar ambas partes o que sirven de complemento para alguna de ellas.

4.1 BLOCKCHAIN

Comenzamos con el primer apartado, que incluye todo lo referente a la cadena de bloques. Para el desarrollo de aplicaciones descentralizadas en Ethereum, este provee a los desarrolladores de algunas herramientas que facilitan el trabajo de pruebas y despliegue.

4.1.1 SMART CONTRACTS

Los contratos inteligentes se han desarrollado gracias al lenguaje de programación Solidity, orientado a contratos y creado específicamente para usarse en la EVM. Cada contrato representaría una clase en un lenguaje de programación orientado a objetos, y en su interior, es posible declarar diferentes estructuras de datos como arrays o mappings, e implementar funciones y eventos.

Antes de comentar los contratos implementados, cabe destacar las formas en las que Solidity almacena datos:

- **Storage:** Se guardan variables de estado, el almacenamiento es persistente dentro de las estructuras de datos del smart contract, es el tipo por defecto para variables locales.
- **Memory:** Almacenamiento no persistente, es el tipo por defecto para los parámetros de función.
- **Calldata:** No es modificable ni persistente, los datos se comportan como memoria.

Para el presente proyecto se ha utilizado la versión 0.8.0 de Solidity, tratándose en el momento actual de una de las versiones estables más recientes. Se han desarrollado dos contratos inteligentes, “Auth.sol” y “Properties.sol”, los cuales estarán almacenados en el directorio contracts y contendrán la lógica de DApp:

- **Auth:** Tiene el propósito de validar a los usuarios en la plataforma. Permite registrar, loguear y desloguear usuarios.
- **Properties:** Permite a los usuarios logueados publicar propiedades, comprarlas, venderlas o alquilárlas. En caso de ser un usuario titular de una propiedad que no ha sido vendida o alquilada, tiene la posibilidad de eliminarla de la plataforma, para así remediar posibles errores durante el proceso de publicación.

Para mantener el formato del documento, la implementación detallada se muestra en el Anexo 1, el cual se recomienda revisar encarecidamente para poder comprender todas las funcionalidades de los smart contracts desarrollados.

4.1.2 TRUFFLE

Los contratos inteligentes especificados en el apartado anterior, han sido desarrollados, compilados y testeados utilizando el framework Truffle [15], el cual ofrece la posibilidad de configurar una red, o cadena de bloques, de Ethereum en nuestro entorno local de desarrollo, desde la consola de comandos, lo cual será vital, puesto que un despliegue en la red principal de Ethereum, no podría ser revertido, y la aplicación quedaría ya desplegada para siempre.

En una blockchain de desarrollo en local, podríamos decir que somos el único nodo, o participante de la red, así como los propietarios de la misma. Sin embargo, es el entorno idóneo para desarrollar y testear nuestros smart contracts, antes de desplegarlos en una red real.

Para configurar una red de desarrollo, habrá que ejecutar el comando “truffle init”, el cual creará los directorios contracts, migrations y test, además del fichero truffle-config.js, el cual permite configurar diferentes redes.

```
module.exports = {
  networks: {
    monpaddev: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*",
      gas: 5000000
    }
  };
}
```

Figura 24: Configuración red de desarrollo en truffle-config.js
Fuente: Elaboración propia

En el fichero de configuración se especifican las redes a configurar, en nuestro caso se ha configurado una red llamada “monpaddev”, dónde el host es el equipo que estamos utilizando y el puerto es el 7545. El campo network_id hace referencia a las redes sobre las que se pueden desplegar los contratos, marcando un asterisco le indicamos que puede escoger cualquier red. Finalmente, el campo gas, indica el máximo de gas a utilizar para cada transacción.

El directorio contracts almacena los contratos inteligentes desarrollados, mientras que el directorio migrations contiene ficheros de migraciones, los cuales permiten desplegar los contratos desarrollados a la red de desarrollo mediante el comando “truffle migrate” o “truffle deploy”, una vez hayan sido compilados previamente, ejecutando el comando “truffle compile”.

El aspecto de un fichero de migraciones es el siguiente:

```
var Properties      = artifacts.require('../contracts/Properties.sol');
var Auth            = artifacts.require('../contracts/Auth.sol');

module.exports = function(deployer){
  deployer.deploy(Properties);
  deployer.deploy(Auth);
}
```

Figura 25: Configuración fichero de migraciones de Truffle
Fuente: Elaboración propia

Truffle también permite escribir tests en Solidity o en JavaScript [16] y ejecutarlos, junto con otros frameworks de JavaScript, como Chai [17], para posteriormente ejecutarlos en consola, estos se almacenarán en el directorio test.

Vamos a mostrar un ejemplo sencillo de un test, en el cual se despliega el contrato Properties y se verifica que se ha desplegado satisfactoriamente comprobando la dirección del smart contract creado:

```
const Properties = artifacts.require('Properties');
const { assert } = require("chai");

contract('Properties', () => {
  before(async() => {
    this.Properties = await Properties.deployed();
  });

  it('Properties contract deployed successfully', async() => {
    const address = this.Properties.address;

    assert.notEqual(address, null);
    assert.notEqual(address, undefined);
    assert.notEqual(address, 0x0);
    assert.notEqual(address, "");
    console.log(address);
  });
});
```

Figura 26: Implementación de test desarrollado en JavaScript
Fuente: Elaboración propia

La ejecución del test se realiza simplemente escribiendo “truffle test”, dónde Truffle compilará y ejecutará todos los archivos que haya en el directorio test y devolverá errores si los hubiera.

El uso de tests es una herramienta muy útil a la hora de hacer pruebas de los contratos inteligentes durante las fases de desarrollo, junto con la otra funcionalidad que nos proporciona Truffle, la consola, a la que se puede acceder ejecutando el comando truffle console.

4.1.3 GANACHE

Una vez tenemos Truffle configurado, empezamos a hacer uso de Ganache [18], dónde la blockchain de desarrollo local “monpadev” configurada, la podemos pasar a un entorno gráfico, simplemente añadiendo los datos de la red configurada en el fichero truffle-config.js:

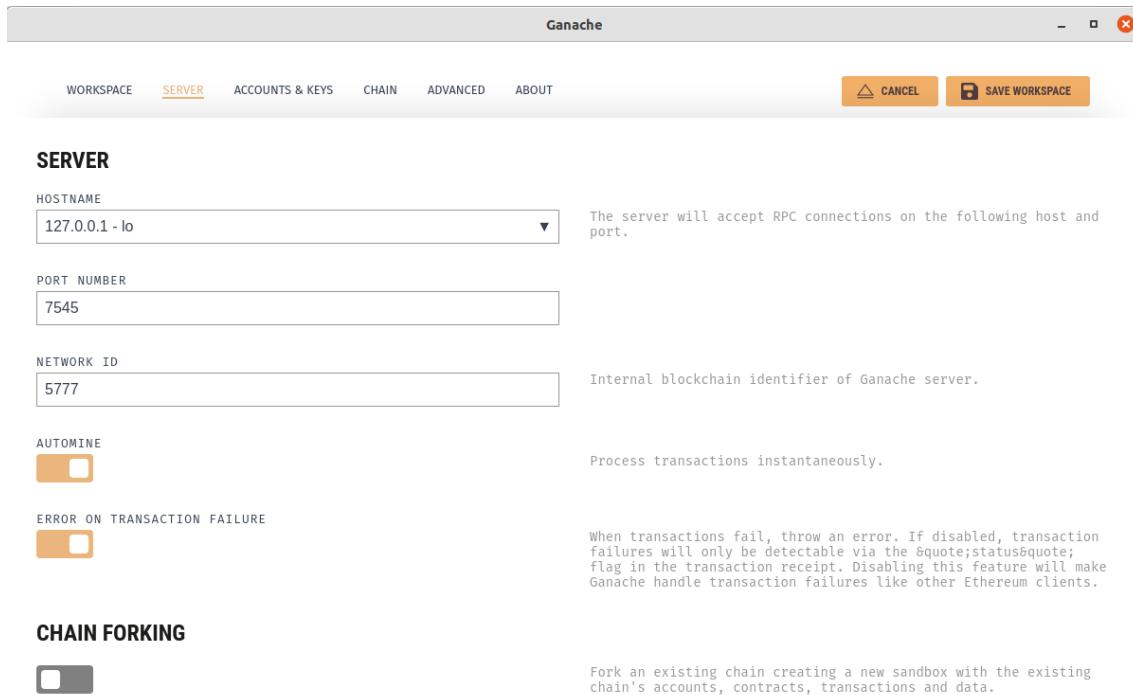
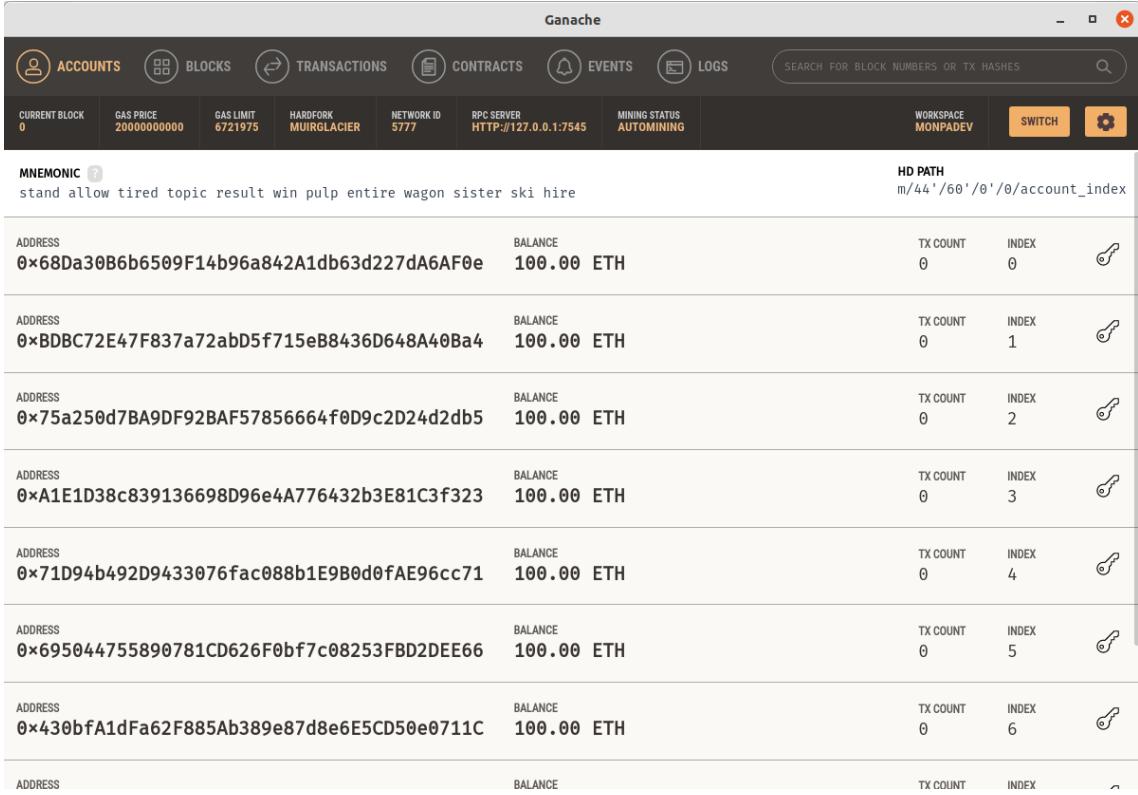


Figura 27: Configuración de Ganache
Fuente: Elaboración propia

Una vez introducidos los datos de configuración, tenemos la red local de desarrollo configurada en el entorno gráfico de Ganache, desde el cual vemos que Truffle nos provee de una billetera con 10 cuentas con 100 Ethers de testing en cada una de ellas:



The screenshot shows the Ganache graphical interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below these are settings for CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK (MUIRGLAGIER), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). The WORKSPACE is set to MONPADEV. There are buttons for SWITCH and a gear icon. A search bar at the top right allows searching for block numbers or tx hashes.

MNEMONIC	HD PATH
stand allow tired topic result win pulp entire wagon sister ski hire	m/44'/60'/0'/0/account_index
ADDRESS 0x68Da30B6b6509F14b96a842A1db63d227dA6AF0e	BALANCE 100.00 ETH TX COUNT 0 INDEX 0
ADDRESS 0xBDBC72E47F837a72abD5f715eB88436D648A40Ba4	BALANCE 100.00 ETH TX COUNT 0 INDEX 1
ADDRESS 0x75a250d7BA9DF92BAF57856664f0D9c2D24d2db5	BALANCE 100.00 ETH TX COUNT 0 INDEX 2
ADDRESS 0xA1E1D38c839136698D96e4A776432b3E81C3f323	BALANCE 100.00 ETH TX COUNT 0 INDEX 3
ADDRESS 0x71D94b492D9433076fac088b1E9B0d0fAE96cc71	BALANCE 100.00 ETH TX COUNT 0 INDEX 4
ADDRESS 0x695044755890781CD626F0bf7c08253FBD2DEE66	BALANCE 100.00 ETH TX COUNT 0 INDEX 5
ADDRESS 0x430bfA1dFa62F885Ab389e87d8e6E5CD50e0711C	BALANCE 100.00 ETH TX COUNT 0 INDEX 6
ADDRESS	BALANCE TX COUNT INDEX

Figura 28: Entorno gráfico de Ganache
Fuente: Elaboración propia

Como podemos observar en la parte superior, vemos el nombre del espacio de trabajo “monpaddev”, así como el host y el puerto, y las 10 cuentas con sus direcciones de clave pública.

Las palabras que vemos al inicio, forman el llamado mnemónico, el cual corresponde a la entrada de datos que forman la clave privada. En un entorno real, estas palabras jamás deberían estar visibles, ya que cualquiera podría acceder a nuestra billetera y por ende, a nuestros fondos.

También hay que destacar la importancia de los menús superiores, dónde se puede ver al detalle información esencial, cómo los bloques minados de la red, las transacciones, los contratos desplegados o los eventos emitidos:

DApp de compraventa y alquiler de inmuebles
Héctor Montesinos

Ganache					
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS
CURRENT BLOCK 1888	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MURKELAICEN	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545
					Mining Status AUTOMINING
BLOCK 1888	MINED ON 2022-01-23 16:46:08				GAS USED 51557
BLOCK 1887	MINED ON 2022-01-23 16:45:42				GAS USED 553899
BLOCK 1886	MINED ON 2022-01-23 16:45:42				GAS USED 83386
BLOCK 1885	MINED ON 2022-01-23 16:44:43				GAS USED 51557
BLOCK 1884	MINED ON 2022-01-23 16:40:23				GAS USED 3128169
BLOCK 1883	MINED ON 2022-01-23 16:40:21				GAS USED 4551232
BLOCK 1882	MINED ON 2022-01-23 16:46:51				GAS USED 32357
BLOCK 1881	MINED ON 2022-01-23 12:42:38				GAS USED 122742
BLOCK 1880	MINED ON 2022-01-23 11:53:48				GAS USED 32357
BLOCK 1879	MINED ON 2022-01-23 11:36:52				GAS USED 51557
BLOCK 1878	MINED ON 2022-01-23 11:32:14				GAS USED 553711
BLOCK 1877	MINED ON 2022-01-23 11:32:14				GAS USED 83386
BLOCK 1876	MINED ON 2022-01-23 11:31:10				GAS USED 83437
BLOCK 1875	MINED ON 2022-01-23 11:03:46				GAS USED 547442
BLOCK -- -- -	MINED ON				GAS USED 83389

Figura 29: Visualización de bloques minados en Ganache

Fuente: Elaboración propia

Lo que observamos en la “figura 29”, es de las configuraciones menos realistas de Ganache, dónde sólo se mina un bloque cuando hay una transacción, a diferencia de la red real de Ethereum, dónde se mina un bloque cada 15 segundos, y cada uno de ellos contiene miles de transacciones. Sin embargo, Ganache nos provee de la posibilidad de configurar una red local, en la cual probablemente trabajen unos pocos usuarios como máximo.

Ganache					
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS
CURRENT BLOCK 1888	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MURKELAICEN	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545
					Mining Status AUTOMINING
TX HASH 0x1434fbdd8247a15cf1b6d5a06e6da7ff69f282695a342bb469994dcfed92c1ba	FROM ADDRESS 0x53bd8ca1f533536cA8110cF0cF3b0b642e18134	TO CONTRACT ADDRESS Auth			GAS USED 51557 VALUE 0
TX HASH 0x3d06d373ea8c6c8b5dc543d8311398e6780e19fa47dc56f6588b9e155fe6db3f	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	TO CONTRACT ADDRESS Properties			GAS USED 553899 VALUE 0
TX HASH 0x31ea529233b2cf88c13705bc7bc07a8225208db4de3c2cdf6ce00b1aadc796b	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	TO CONTRACT ADDRESS Properties			GAS USED 83386 VALUE 0
TX HASH 0x5a72ed5feb20ba0526b3b9655e6b64e61da3eb0b7d353f0b28f7731f2dd11fe0	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	TO CONTRACT ADDRESS Auth			GAS USED 51557 VALUE 0
TX HASH 0x2cf95e002297fb4b560e09f9e04cb0f4c9b3c44e19aa1ba35778936ca977b814	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	CREATED CONTRACT ADDRESS 0x9c907c9c9a5b7d5e8f9e88f54cd464bf740e9b905			GAS USED 3128109 VALUE 0
TX HASH 0xa5c154578d5870ece93c7fc7c6f0a60fa14b3cae7a26962cf2e1cec85b949e87f7	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	CREATED CONTRACT ADDRESS 0x3c2751d8888922c2ec1f488c3f9c24e80d6bc4e			GAS USED 4551232 VALUE 0
TX HASH 0xa5c154578d5870ece93c7fc7c6f0a60fa14b3cae7a26962cf2e1cec85b949e87f7	FROM ADDRESS 0xc43cd2ff3fc1a6f3f8b785659b71687358562335	CREATED CONTRACT ADDRESS 0x3c2751d8888922c2ec1f488c3f9c24e80d6bc4e			GAS USED 4551232 VALUE 0

Figura 30: Visualización de transacciones en Ganache

Fuente: Elaboración propia

Se pueden ver todas las transacciones sucedidas, dónde cada una incluye su hash de transacción, la dirección de clave pública que la ha emitido, el contrato inteligente que recibe o, si se trata del despliegue de un contrato inteligente.

Haciendo clic sobre cada transacción, se puede ver un detalle de la misma:

Figura 31: Visualización detalle de transacción en Ganache
Fuente: Elaboración propia

Dentro de cada transacción se pueden ver detalles como el bytecode, si se ha transferido un balance en dicha transacción, el bloque en el cual ha sido minada, el contrato inteligente que la incluye y los eventos emitidos en ella.

Ganache

ACCOUNTS BLOCKS TRANSACTIONS CONTRACTS EVENTS LOGS

CURRENT BLOCK GAS PRICE GAS LIMIT HARDFORK MURGLACIER NETWORK ID RPC SERVER MINING STATUS AUTOMINING

BACK Properties

ADDRESS 0x3c2751d888922C2eec1E48bcf3f9c24e86D6BC4e BALANCE 0.00 ETH

CREATION TX 0x0c154578d5870Ee93C7fc7c6f0ab0fA14B3CAE7A26962CF2e1cEC85B949E87F

STORAGE

```
contracts/Property.sol:Property:storage
```

```
0x3c2751d888922C2eec1E48bcf3f9c24e86D6BC4e
```

```
|- 0: { ... } 2 items
  |- 1: { ... } 2 items
    type : string "Property"
    members : { 11 items
      id : string "19-HN0001-0001-SC"
      owner : address "0xc3b2cf23f164f318..."
      city : string "Sant Boi de Llobregat"
      physicalAddr : string "Carrer de la Mort, 1, ..."
      price : uint 2221736755280000
      rooms : uint 3
      area : uint 64
      bathrooms : uint 2
      sellerRent : uint 1
      createdAt : uint 62537745
      soldOn : uint 62540461
    }
  |- 2: { ... } 2 items
  |- 3: { ... } 2 items
cnt : uint 4
cntTokens : uint 0
```

Figura 32: Visualización de contrato inteligente desde Ganache
 Fuente: Elaboración propia

También es posible ver al detalle los smart contracts desplegados en la red, así como la información que incluyen, por ejemplo, en la “figura 32” se aprecian los datos de una propiedad, dentro del contrato inteligente “Properties”, así como de las estructuras de datos incluidas en su interior.

4.1.4 ORÁCULOS Y REDES

Cómo ya se ha comentado anteriormente, los smart contracts viven dentro de la blockchain, y, por lo tanto, sólo manejan información en ese entorno, lo que se conoce como entorno “on-chain”.

En ocasiones es necesario hacer uso de oráculos, para permitir a los contratos inteligentes consultar información del exterior, en el caso particular de este proyecto, se consultarán datos del exterior para dos razones concretas:

- **Consultar el precio del Ether:** Los envíos de divisas dentro de una blockchain pública se realizan mediante el token nativo de su red, en el caso de Ethereum, mediante el Ether (token ETH). El Ether es volátil, y contra divisas tradicionales como el euro o el dólar, fluctúa constantemente, por lo que es necesario consultar en cada momento su valor, para mantener el precio que los usuarios fijan a la hora de publicar propiedades.
- **Verificar el registro de la propiedad:** El registro de la propiedad es una institución del estado español, donde se inscriben los inmuebles para saber quién es el propietario, y sus derechos y cargas sobre una propiedad. Para aplicar la máxima seguridad, sólo se permitirá la publicación de propiedades a los usuarios hacia los cuales se haya verificado la titularidad de las mismas.
En este caso, es necesario que, en el momento de la publicación, se compruebe que el par *<Referencia catastral, Datos usuario>* introducido, coincida con los datos existentes en el registro de la propiedad vigente.

Cómo oráculo se quiere hacer uso de Chainlink [19], un proyecto de oráculo descentralizado, desarrollado como un middleware, el cuál puede acceder a los contratos inteligentes y a sistemas externos.

Sin embargo, existen algunos problemas en este punto con los que nos hemos encontrado, y es que, Chainlink no da soporte a redes locales de desarrollo, como por ejemplo a la red de Truffle, simplemente da soporte a Ethereum y otras blockchains y sus redes reales.

Esto podía suponer un problema, ya que en nuestro proyecto es necesario que nos conectemos con el exterior de la red para obtener datos como hemos visto en los dos puntos anteriores. Sin embargo, Ethereum, además de contar con su red principal o “Mainnet”, cuenta con redes de prueba públicas o “Testnets”, las cuales están creadas precisamente para desarrollo, en las cuales Chainlink u otros oráculos sí que dan soporte.

Para acceder a alguna de las redes públicas, tenemos dos opciones:

1. Disponer de un nodo de Ethereum: Formar parte de la red dando soporte a la cadena de bloques, mediante la descarga del software indicado.
2. Utilizar un producto IaaS (Infraestructure as a Service) como Infura [20], el cual permite acceder a la red de Ethereum sin necesidad de ejecutar un nodo localmente: En ocasiones el despliegue de un nodo local puede ser innecesario si el objetivo no es convertirse en validador de la red, además del coste que puede suponer energéticamente y de mantenimiento.

En el presente proyecto se va a hacer uso de Infura, ya que la instalación de un nodo de Ethereum supone tener un equipo con unos requisitos mínimos bastante elevados [21] y costosos, ya que, para participar en la red, hay que descargar todo el histórico de transacciones de la cadena de bloques hasta la fecha, la cual, a 20 de enero de 2022, tiene un tamaño de 1178.68 GB [22], por tanto, sumando al espacio libre que se recomienda dejar para poder almacenar el contenido de la red conforme va aumentando, solamente en almacenamiento de disco ya supone un coste muy elevado.

El entorno de Infura permite crear un proyecto en la blockchain de Ethereum, entre otras redes:

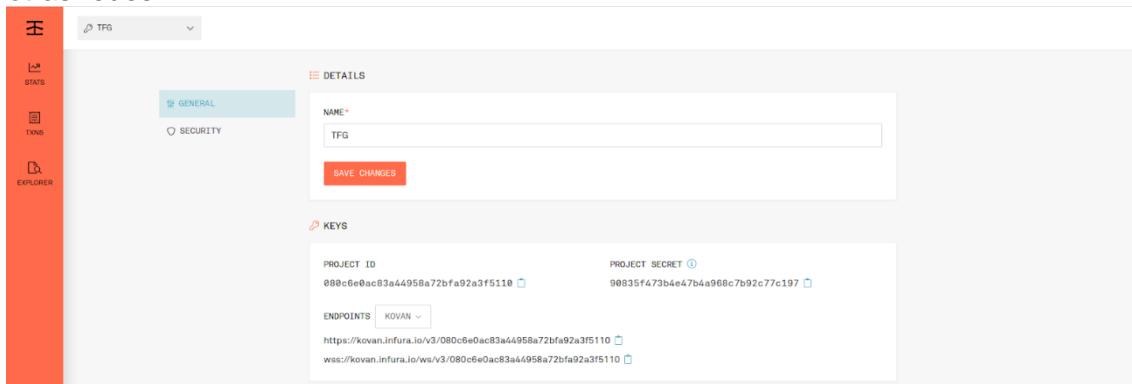


Figura 33: Interfaz de Infura
Fuente: Elaboración propia

Una vez se ha creado el proyecto en Ethereum, este se puede conectar a una de sus redes, ya sea la red principal o cualquiera de sus redes de pruebas. La red principal queda descartada, puesto que cada transacción en Ether, tendrá un coste real en gas para nosotros y queda fuera del alcance de la prueba de concepto que queremos representar con este proyecto. Por lo tanto, nos decantamos por la red de pruebas “Kovan”, dónde los Ether que se usarán serán de testing, pese a que, a simple vista serán idénticos a los reales.

Cabe destacar que, a diferencia de Ganache, Kovan no provee a los usuarios 100 ETH de balance por cuenta, al tratarse de una red de pruebas pública, por lo tanto, hay que acceder a páginas webs, llamadas faucets [23], que proveen a los usuarios de Ethers de testing, similarmente a lo que hace automáticamente Ganache, pero en este caso los usuarios tendrán que añadir las claves públicas de sus cuentas para que los Ethers de testing sean enviados a sus cuentas. En nuestro caso, vamos a utilizar un faucet que provee Chainlink en su sitio web oficial [24].

En la “figura 33” se puede observar cómo Infura provee de un id de proyecto, así como de unos endpoints, para poder conectarnos con su infraestructura desde el código, lo cual se realiza desde el fichero de configuración de Truffle, el cual ya conocemos.

Otro punto importante a la hora de realizar la conexión es el mnemónico, ya que también se tendrá que utilizar para conectar la billetera que ya poseemos en Ganache. Para la configuración se requiere importar el módulo HDWalletProvider [25], el cual provee a la aplicación de una billetera para añadir cuentas:

```
const HDWalletProvider = require('truffle-hdwallet-provider');
const mnemonic = 'fun garment pudding day love other color bulk
clarify have honey physical';

module.exports = {
  networks: {
    monpadev: {
      // ...
    },
    kovan: {
      provider: () => new HDWalletProvider(mnemonic,
"https://kovan.infura.io/v3/080c6e0ac83a44958a72bfa92a3f5110"),
      network_id: 42
    }
  }
}
```

Figura 34: Configuración de red Kovan en truffle-config.js

Fuente: Elaboración propia

Para no repetir código, se ha ocultado la configuración de la ya conocida red local “monpadev”, la cual no ha cambiado desde la última captura. La configuración de la red Kovan incluye el proveedor, que lo forman el mnemónico de la billetera y el endpoint que anteriormente nos proporcionó Infura, y el id de red, el cual por decreto es el 42 [26]. Una vez configurada la red Kovan, el proyecto se desplegará mediante el comando “truffle migrate --network kovan”, para diferenciarlo con el resto de redes configuradas.

Llegados a este punto, y después de haber realizado tests en la infraestructura de Infura y ver su potencial, nos encontramos con algunos contratiempos. En primer lugar, Infura es un servicio centralizado, lo cual choca con el principio de descentralización que poseen las cadenas de bloques públicas, además, hay un cierto límite en el número de transacciones diarias, y si este es sobrepasado, es necesario adquirir una tarifa de pago, lo cual supone un problema para el desarrollo del proyecto.

Por este motivo, nos encontramos con problemas para llevar a cabo tanto el presente apartado como el siguiente apartado de tokenización, ya que la red local de Truffle, no es capaz de conectarse con el exterior.

La solución escogida es, mostrar el uso de oráculos en la red de pruebas Kovan utilizando Infura, y posteriormente, de cara al resto de pruebas y al producto final, mostrar el proyecto en su totalidad en Ganache, implementando soluciones alternativas al problema anterior.

Volviendo al uso de oráculos y concretamente de Chainlink, este provee a los desarrolladores de algunas fuentes de datos, algunas de ellas las referentes a conversiones de divisas en tiempo real [27], lo cual es de gran utilidad para uno de los requerimientos de nuestro proyecto.

Gracias a estos datos, es posible obtener una conversión entre divisas y criptodivisas rápidamente, para ello, hemos implementado un smart contract llamado PriceConverter.sol, el cual no se ha añadido al apartado smart contracts ya que como hemos comentado anteriormente, el resto del proyecto se desarrollará en la red local de Ganache, y en este apartado sólo se realizarán demostraciones.

Para desarrollar este contrato inteligente ha sido necesario instalar un módulo de Chainlink, el cual importará funciones internas. El contenido es el siguiente:

```
contract PriceConverter {
    AggregatorV3Interface internal eth_usd;
    AggregatorV3Interface internal eur_usd;
    constructor() {
        eth_usd = AggregatorV3Interface(0x9326BFA02ADD2366b30bacB125260Af641031331);
        eur_usd = AggregatorV3Interface(0x0c15Ab9A0DB086e062194c273CC79f41597Bbf13);
    }

    function getEthUsd() public view returns (int) {
        (
            uint80 roundID,
            int price,
            uint startedAt,
            uint timeStamp,
            uint80 answeredInRound
        ) = eth_usd.latestRoundData();

        return price;
    }

    function getEurUsd() public view returns (int) {
        (
            uint80 roundID,
            int price,
            uint startedAt,
            uint timeStamp,
            uint80 answeredInRound
        ) = eur_usd.latestRoundData();

        return price;
    }

    function getEthEur(uint _amountInEur) public view returns (uint) {

        uint newInput = _amountInEur * 10 ** 18;
        uint EthUsd = uint(getEthUsd());
        uint EurUsd = uint(getEurUsd());

        return newInput * EurUsd / EthUsd;
    }
}
```

Figura 35: Implementación smart contract PriceConverter.sol
Fuente: Elaboración propia

El contrato simplemente llama a las fuentes de datos EthUsd y EurUsd, ya que la fuente de datos EurEth no existe, y posteriormente, dada una cifra en euros introducida por el usuario, la convierte a Wei y la multiplica por el resultado de la división de las fuentes de datos comentadas.

A continuación, y de forma breve, demostraremos el correcto funcionamiento del smart contract de la “figura 35” desde la consola de Truffle usando la red de pruebas Kovan, calculando el valor actual en Ether para 1000€:

```
truffle(kovan)> contract = await PriceConverter.deployed();
undefined
truffle(kovan)> eurToWei = await contract.getEthEur(1000);
undefined
truffle(kovan)> eurToWei
BN {
  negative: 0,
  words: [ 45790087, 37010624, 77, <1 empty item> ],
  length: 3,
  red: null
}
truffle(kovan)> weiToEth = web3.utils.fromWei(eurToWei, 'ether');
'0.349260912285889415'
```

Figura 36: Conversión EUR a ETH desde la consola de Truffle
Fuente: Elaboración propia

Cómo vemos en la “figura 36”, para realizar el cálculo se ha desplegado el contrato inteligente desde la consola de Truffle y se ha hecho uso de la librería web3.utils [28], que integra Truffle.

Para finalizar la comprobación, desde la consola también podemos comprobar que bloque se está minando en cada momento, y por tanto, obtener timestamp:

```
truffle(kovan)> blockNum = await web3.eth.getBlockNumber();
undefined
truffle(kovan)> blockNum
29366116
truffle(kovan)> block = await web3.eth.getBlock(blockNum);
undefined
truffle(kovan)> block['timestamp'];
1642695288
```

Figura 37: Obtención bloque actual desde la consola de Truffle
Fuente: Elaboración propia

Al obtener el timestamp del bloque, ya que la blockchain de Ethereum utiliza el sistema horario de UNIX, podemos revisar la fecha a la que corresponde desde cualquiera consola Linux:

```
hector@Ubuntu-21:~/Sites$ date -d @1642695288
jue 20 ene 2022 17:14:48 CET
```

Figura 38: Obtención de fecha mediante timestamp en consola de Linux
Fuente: Elaboración propia

Y finalmente, comprobamos en cualquier conversor de criptodivisas, en la fecha y hora indicada por el Timestamp, que la conversión realizada desde la consola Truffle es correcta. Cada 1000€ corresponden aproximadamente a 0.349 Ether, o lo que es lo mismo, cada Ether tiene un valor aproximado de 2865€, a 20 de enero de 2022:

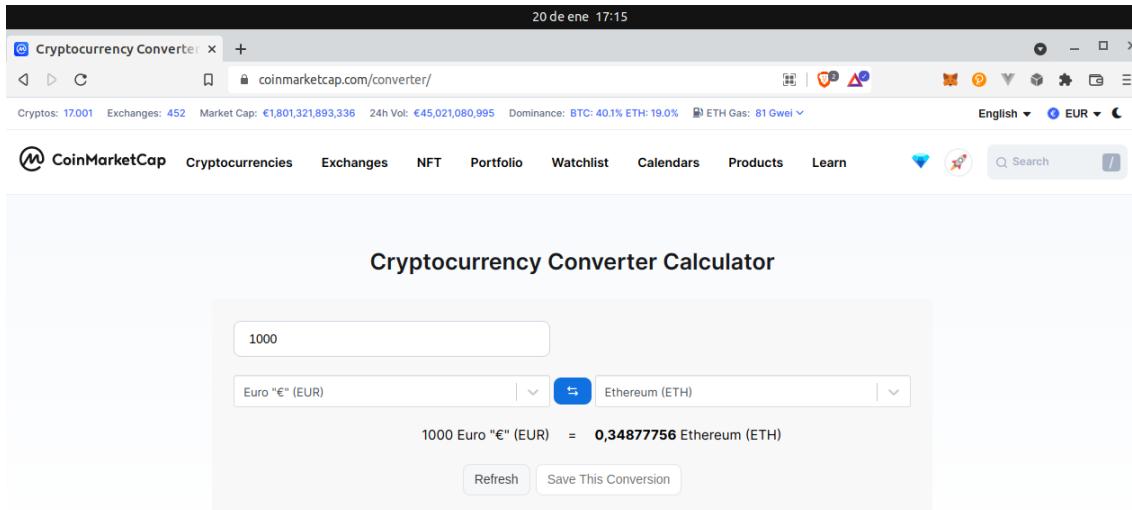


Figura 39: Conversión EUR a ETH desde conversor web de criptodivisas
Fuente: coinmarketcap.com

4.1.5 TOKENIZACIÓN

Un usuario tiene la posibilidad de alquilar propiedades de manera tokenizada, si un propietario alquila su propiedad utilizando esta funcionalidad. El propósito de la plataforma, es acuñar (lo actualmente se conoce como “mintear”) un token ERC-721, o lo que es lo mismo, un NFT, para cada token que un usuario adquiera, a modo de entregarle un objeto digital que simula su fracción en la propiedad que está alquilando.

Sin embargo, debido a los problemas comentados en el apartado anterior, no sé podrá llevar a cabo, ya que como se ha comentado anteriormente, no se hará uso de la infraestructura de Infura, salvo en pruebas de testing puntuales, pero sí que se les mostrará a los usuarios los tokens que poseen, además de que podrán verse en el explorador de la cadena de bloques, mediante Ganache, como mostraremos más adelante.

No obstante, y ya que se ha investigado acerca de ello, se va a mostrar cual sería el procedimiento a seguir para mintear un NFT, mediante un contrato inteligente y la tecnología de IPFS, dos utilidades que, llegados a este punto, ya conocemos.

Gracias a librerías que provee Solidity, es posible crear un nuevo smart contract a partir de la importación de otros contratos para utilizar sus funciones:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "https://github.com/0xcert/ethereum-erc721/src/contracts/tokens/nf-token-metadata.sol";
import "https://github.com/0xcert/ethereum-erc721/src/contracts/ownership/ownable.sol";

contract mintNFT is NFTokenMetadata, Ownable {
    constructor() {
        name = "Hmonpa NFT";
        symbol = "MONPA";
    }
    function mint(address _receiver, uint256 _tokenId, string _file) external {
        _mint(_receiver, _tokenId);
        _setTokenUri(_tokenId, _file);
    }
}
```

Figura 40: Implementación smart contract mintNFT.sol

Fuente: Elaboración propia

En el smart contract anterior, le damos un nombre y un símbolo al token no fungible al inicializar el constructor. Posteriormente, se declara la función “mint”, la cual almacena la dirección de clave pública del receptor del NFT, su ID y un fichero que almacenará la información personalizada para cada poseedor. Dicho fichero, se almacenará previamente en IPFS, por lo que, la información que contiene este campo, es su hash CID.

Este podría ser un ejemplo del fichero en cuestión:

```
hector@Ubuntu-21:~/Sites$ cat nft.json
{
    "name": "Monpa NFT",
    "description": "Inmueble con referencia catastral XXX-XXX-XXXXXX",
    "image": "https://ipfs.io/ipfs/QmWdQdQcCSfrWwReRekR3NnyrHy47nfBra7Qk8Dx8zjjHs"
}
```

Figura 41: Contenido del fichero pasado como argumento en la función mintNFT

Fuente: Elaboración propia

El fichero en cuestión, contiene como información un nombre, una descripción y hash CID de una imagen, también almacenada previamente en IPFS.

Posteriormente, desplegamos el contrato y ejecutamos la función mint desde la consola de Truffle, en la red Kovan, dónde pasaremos como parámetros la clave pública de la primera de las 10 cuentas que nos provee Truffle, la cual recibirá el NFT, después, el ID del token, en este caso 1, al ser el primer NFT que se va a mintear, y, por último, el enlace IPFS con el hash CID del fichero mostrado en la captura anterior:

```
truffle(kovan)> contract = await mintNFT.deployed();
undefined
truffle(kovan)> mint = await contract.mint(0xc43cb2ff3FC1A6f3f8B785659b71687350562335, 1,
"https://ipfs.io/ipfs/QmZkHrLPdzPY187T8kaHDXUBYB34gRU2aLoQgc1hb2jckD");
undefined
truffle(kovan)> mint['tx']
'0x3532b3deb3d2049be3ff7509af86a9c1f46524866f5b2ceb24f7fc8ef9527e20'
```

Figura 42: Generación de NFT desde la consola de Truffle

Fuente: Elaboración propia

Tras la ejecución de la función mint, hemos obtenido el hash de la transacción, gracias a ello, ahora podemos acceder a Etherscan [29], un explorador para la blockchain de Ethereum, en el cual podemos ver al detalle transacciones, bloques, direcciones de clave pública, contratos inteligentes y demás información del entorno on-chain, para todas sus redes.

Simplemente añadiendo copiando el hash de la transacción podemos obtener toda su información:

The screenshot shows the Etherscan interface for a Kovan Testnet transaction. At the top, there's a navigation bar with the Etherscan logo, a dropdown for 'All Filters', and a search bar. Below the header, it says 'Kovan Testnet Network'. The main area is titled 'Transaction Details' with tabs for 'Overview', 'Logs (1)', and 'State'. A note '[This is a Kovan Testnet transaction only]' is displayed. The transaction details include:

- Transaction Hash: 0x3532b3deb3d2049be3ff7509af86a9c1f46524866f5b2ceb24f7fc8ef9527e20
- Status: Success
- Block: 29366583 | 2027 Block Confirmations
- Timestamp: 2 hrs 37 mins ago (Jan-20-2022 04:48:40 PM +UTC)
- From: 0xc43cb2ff3fc1a6f3f8b785659b71687350562335
- Interacted With (To): Contract 0x7bcd72e9914f75bcad10307a85e72757c9ace818
- Tokens Transferred: From 0x0000000000000000 To 0xc43cb2ff3fc1a6f3f8b785659b71687350562335 For ERC-721 TokenID [1] Hmonpa NFT (MONPA)
- Value: 0 Ether (\$0.00)
- Transaction Fee: 0.000408127501142757 Ether (\$0.00)
- Gas Price: 0.00000000250000007 Ether (2.500000007 Gwei)

A 'Click to see More' button is at the bottom.

Figura 43: Transacción generada al ejecutar la función mint en la red Kovan vista en Etherscan
Fuente: kovan.etherscan.io

Siguiendo el principio de inmutabilidad, y de igual manera que los usuarios pueden ver las transacciones realizadas en la red local en Ganache, esta información podrá ser consultada por cualquiera que disponga del hash de la transacción, en cualquier momento y para siempre.

4.2 METAMASK

Metamask es una extensión de navegador, y que también cuenta con aplicación multiplataforma para smartphones, que permite tener una billetera de Ethereum, entre otras redes, que incluya cuentas en su interior. Esto permite a los usuarios interactuar con DApps de la cadena de bloques de Ethereum y poder generar transacciones.

Por defecto tiene configuradas todas las redes de Ethereum, principal y de pruebas, pero también permite configurar diferentes redes, entre las que se incluyen las redes locales de desarrollo, como la que provee Ganache.

Por otro lado, además de tener la posibilidad de crear nuevas billeteras, y cuentas en su interior, también permite importar billeteras existentes, mediante la introducción de mnemónicos, o cuentas existentes, mediante la introducción de sus claves privadas. Una vez importada una billetera o cuenta ya existente, Metamask requiere además de la creación de una contraseña, con el fin de obtener una doble seguridad. Cuando un usuario ya ha importado o creado su billetera, puede añadir nuevas redes. En nuestro caso vamos a configurar la red de desarrollo que ya conocemos:

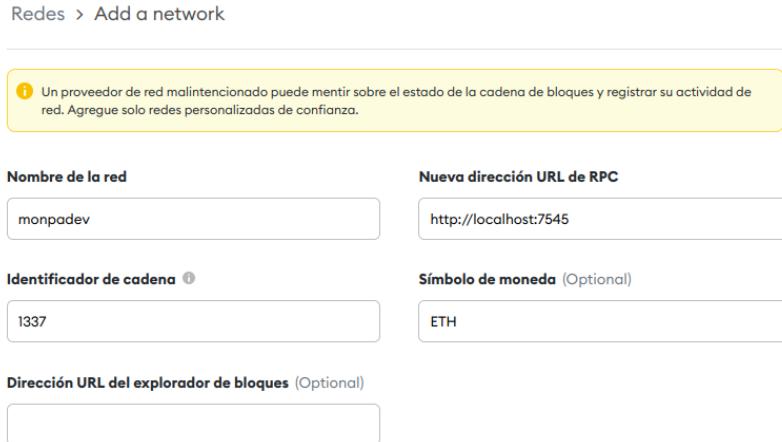


Figura 44: Proceso de configuración red de desarrollo en Metamask
Fuente: Elaboración propia

Cómo se aprecia en la “figura 44”, los datos de configuración de la red son los mismos a los introducidos anteriormente en el fichero “truffle-config.js”, a excepción del identificador de la cadena, el cual por defecto es el 1337 para añadir redes de desarrollo configuradas mediante Truffle [30].

Una vez está importada la billetera, y configurada la red de desarrollo, ya es posible cambiar de red en Metamask, escogiendo nuestra red de desarrollo, y podremos ver el balance de 100 ETH en las 10 cuentas de las que dispone nuestra billetera:

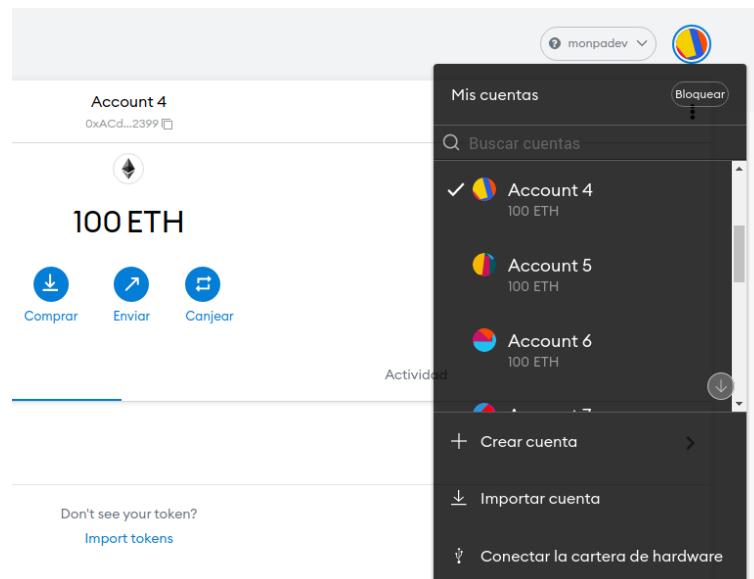


Figura 45: Visualización de billetera Ganache en Metamask en red de desarrollo
Fuente: Elaboración propia

Por último, vamos a mostrar la posibilidad de visualizar los tokens ERC-721, o NFT, en Metamask, y lo haremos con el NFT minteado en el apartado “4.1.5”. Para ello, es necesario cambiar de red en Metamask a la red Kovan, ya que como recordamos, es desde dónde se generó dicho NFT, debido a la imposibilidad de realizar el proceso desde la red de desarrollo.

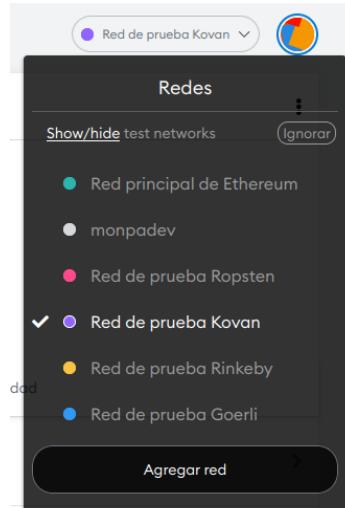


Figura 46: Cambio de red en Metamask y visualización de la nueva red configurada
Fuente: Elaboración propia

Una vez realizado el cambio a la red Kovan, hay que acceder al apartado “import tokens” en Metamask, que se visualiza en la “figura 45” e importar la dirección del contrato mintNFT, la cual podemos encontrar en el campo “Contract” de la transacción en dónde se desplegó el contrato, en el explorador Etherscan en la “figura 43” o también en el apartado actividad de Metamask, dónde aparecen las transacciones realizadas, y nos provee de un enlace dónde podemos ver los detalles de las transacciones en Etherscan:

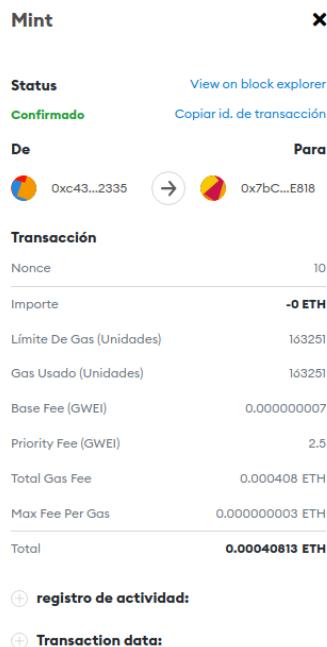


Figura 47: Detalle de una transacción en Metamask
Fuente: Elaboración propia

Una vez obtenida la dirección del contrato inteligente en el campo “Contract” en Etherscan, la pegamos en el apartado “import tokens” dentro del campo “dirección de contrato”:

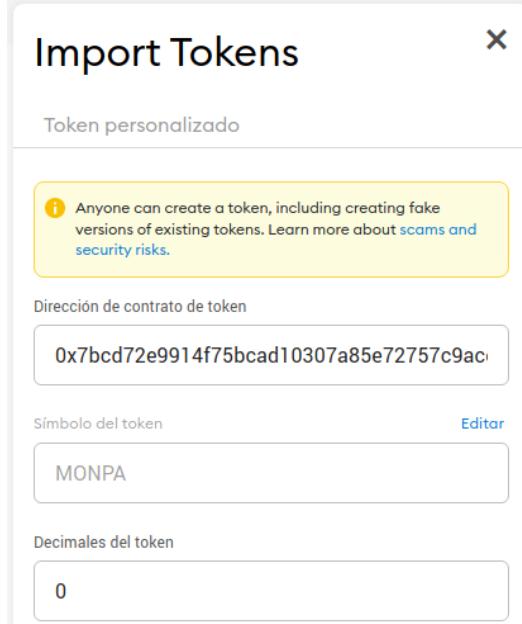


Figura 48: Importación de token ERC-721 en Metamask
Fuente: Elaboración propia

Cómo vemos en la “figura 48”, automáticamente detecta el nombre del token una vez se introduce la dirección del contrato mintNFT desplegado en la red. El único campo que nos pedirá cumplimentar es el del número de decimales, el cual, a diferencia del token ETH del tipo ERC-20, no será divisible en 18 decimales ($1 \text{ ETH} = 10^{18} \text{ Wei}$), ya que los tokens del estándar ERC-721, no son fraccionables, como ya se mencionó al introducir los estándares.

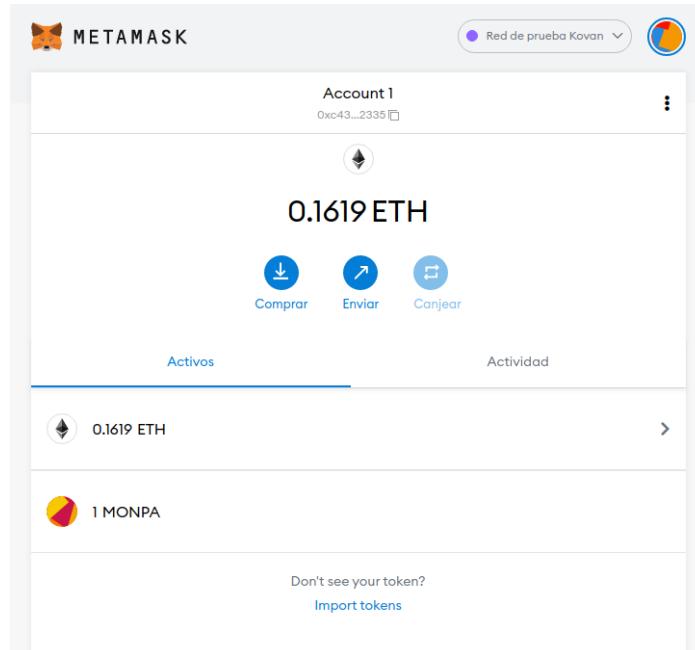


Figura 49: Visualización de token ERC-721 en Metamask
Fuente: Elaboración propia

En la “figura 49”, vemos el token ERC-721, o NFT, en Metamask, junto con el token nativo ETH, del cual el faucet de Chainlink nos proveyó anteriormente de 0.2 ETH, por lo que el gasto computacional en Gas, por mintear el NFT, ha sido de aproximadamente 0.04 ETH.

4.3 FRONT-END WEB

En este punto presentamos la interfaz web, o Front-end, la parte con la que los usuarios interactúan directamente desde un navegador web. Es una parte necesaria de la DApp, debido a qué, el entorno de consola de comandos que provee Truffle es poco intuitivo y apropiado para que los usuarios puedan utilizar cómodamente la plataforma. Para desarrollar el Front-end nos hemos decantado por dos frameworks [31], Vue.js y Nuxt.js.

4.3.1 VUE.JS

Vue.js [32] es un framework escrito en JavaScript, centrado en la parte front-end, es decir, únicamente se centra en la parte de la interfaz de usuario, dejando a un lado el lado del servidor (back-end).

Una de sus características más atractivas es el trabajo con componentes. Cada componente es un elemento que encapsula código de HTML [33], CSS [34] y JavaScript, en un único archivo, lo cual permite crear proyectos fáciles de escalar, pudiendo reemplazar un componente por otro de manera sencilla.

Vue.js se centra en una arquitectura llamada “MVVM” (Model - View - ViewModel), dónde conecta el modelo con la vista:

- **Modelo:** Contiene todo lo relacionado con los datos, ya sea en forma de peticiones de entrada de datos desde el lado del servidor, o en forma de respuestas de salida.
- **Vista:** Contiene la interfaz gráfica que permite al usuario interactuar con la aplicación, es decir, todo el código HTML y CSS.
- **VistaModelo:** En la mayoría de lenguajes esta parte se denomina “Controlador”, pero suele estar más asociado a lenguajes de back-end. Sincroniza el modelo y la vista mediante instancias, las cuales hacen referencia a un objeto principal el cual se inicializa en cada componente.

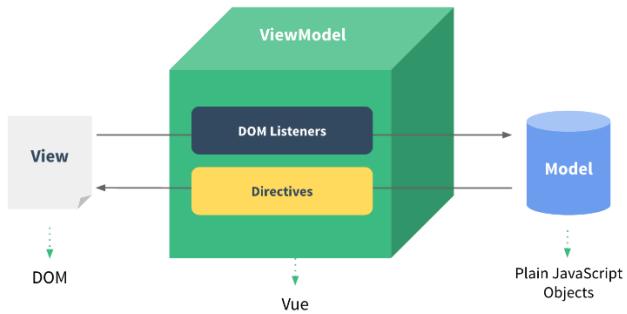


Figura 50: Arquitectura MVVM
Fuente: vuejs.org

4.3.2 NUXT.JS

Nuxt.js [35] es otro framework escrito en JavaScript, y en este caso basado en Vue.js, que cuenta con una estructura de directorios predefinida en el momento de iniciar un proyecto. Utiliza componentes de Vue.js y mejora algunas configuraciones como el enrutamiento entre componentes, lo cual nos servirá de gran ayuda a la hora de conectar los componentes, o pantallas, de nuestra DApp. Nuxt.js.

4.3.3 NODE.JS

Para interactuar con los frameworks anteriores, Node.js [36] provee de un entorno de código abierto que ejecuta código JavaScript en el lado del servidor (back-end), sin necesidad de usar un navegador, lo cual permite intercambiar información con el front-end. Contiene un gestor de paquetes llamado Npm [37], el cual cuenta con librerías que los usuarios pueden descargar con el fin de aplicar funcionalidades específicas a sus proyectos.

4.3.4 WEB3.JS

Web3.js [38] es una librería de JavaScript que permite conectar el front-end de la DApp directamente con los contratos inteligentes desplegados en la blockchain.

Cómo recordamos de la explicación de, que es una cadena de bloques, los nodos que participan en la blockchain de Ethereum, todos tienen una copia actualizada del estado de la red, esta información la reciben a través del protocolo JSON-RPC [39], el cual define varias estructuras de datos.

En este punto es dónde la librería Web3.js desarrolla su gran potencial, ya que permite comunicar el código JavaScript de la DApp directamente con los contratos desplegados en la red.

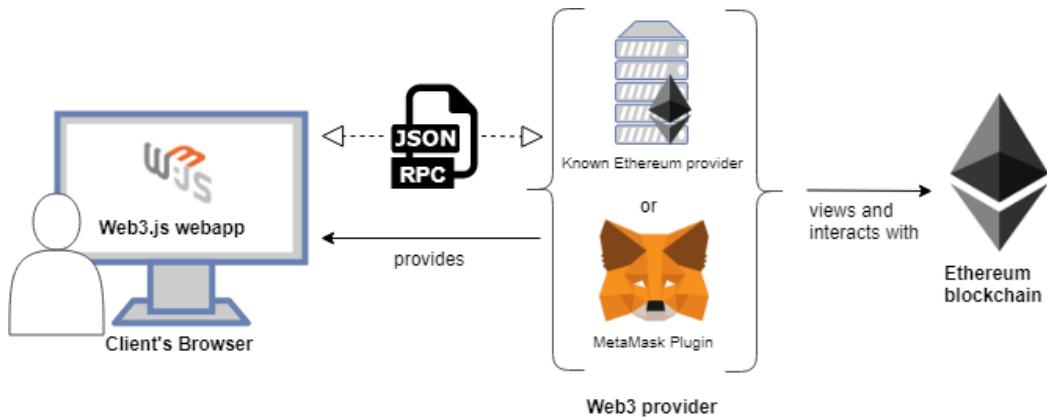


Figura 51: Comunicación mediante Web3.js con la blockchain

Fuente: [medium.com](https://medium.com/@joseantoniofuentes/como-crear-una-dapp-con-ethereum-y-vuejs-5a2a2a2a2a2a)

La conexión entre front-end y smart contracts se realiza desde el fichero dapp.js, el cual contiene todo el código de Node.js, además de librería Web3.js y la librería TruffleContract [40], la cual permite realizar una abstracción de los contratos inteligentes desplegados e interactuar con ellos:

```
var authJson      = require('./build/contracts/Auth.json');
var propertiesJson = require('./build/contracts/Properties.json');
var TruffleContract = require('@truffle/contract');

var Auth      = TruffleContract(authJson);
var Properties = TruffleContract(propertiesJson);
```

Figura 52: Conexión con smart contracts desde Node.js

Fuente: Elaboración propia

4.4 IPFS

En apartados previos ya se introdujo el protocolo InterPlanetary File System, el cual en el presente proyecto se utiliza para almacenar imágenes de inmuebles, así como imágenes de usuario y contratos de compraventa y de alquiler generados por la DApp.

Este procedimiento se realiza desde el front-end de la DApp, para interactuar con IPFS será necesario importar la librería ipfs [41] de Npm, en el componente de Vue.js que vaya a realizar de subida de archivos:

```
import * as IPFS from 'ipfs';

async uploadToIPFS(img)
{
  const node = await IPFS.create();
  let cid = await node.add(img);
  this.ipfsImage = cid.path;
}
```

Figura 53: Subida de imágenes a IPFS desde componente Vue.js

Fuente: Elaboración propia

El procedimiento realizado en el código de la “figura 53” es sencillo, simplemente se conecta a un nodo intermedio en la red IPFS, para después añadir ficheros y obtener su hash CID, el cual servirá para identificarlos posteriormente.

Para la subida de los contratos de compraventa y de alquiler, generados una vez se realiza la compra o el alquiler de un inmueble, el procedimiento es exactamente el mismo, pero en lugar de subir la imagen que el usuario sube a un formulario, se carga una plantilla preparada para la ocasión, dónde aparecen los datos de propietario y comprador, o propietario y arrendatario, respectivamente.

4.5 DATOS EXTERNOS

Cómo vimos en el apartado “4.1.4”, no se ha podido hacer uso de los oráculos desde el front-end, debido a que la red de desarrollo sobre la que corre el proyecto entero, no se comunica con el exterior, y, por tanto, no es posible hacer uso de oráculos. Sin embargo, y como comentamos en ese mismo apartado, se ha buscado una solución a este problema, y se va a comunicar la plataforma con el exterior desde el front-end de la plataforma, lo que obtendrá el mismo resultado que el esperado inicialmente.

Cómo se comentó también en el apartado “4.1.4”, se requiere consultar datos externos para consultar el precio del Ether constantemente, así como para consultar el registro de la propiedad en el momento en que los usuarios publiquen propiedades, para verificar que son los titulares de las mismas.

Para cumplir el primer propósito, se ha creado una API en un servicio web que provee de API's con conversiones de precios de criptodivisas a sus usuarios [42]. Simplemente creando un usuario, nos proveen de una clave privada y la posibilidad de crear API's para diferentes pares de divisas, en nuestro caso, y puesto que la moneda utilizada en nuestro país es el Euro, y la criptodivisa de la red de Ethereum es el Ether, hemos creado una API que nos provea del precio ETH/EUR, la cual consultaremos desde el front-end de la DApp.

La consulta se realiza mediante la librería Axios [43], un cliente HTTP que permite obtener código asíncrono de una URL concreta. En este caso se realiza una solicitud GET, pasando como parámetro la URL de la API en cuestión, y posteriormente, devolviendo el valor del Ether actualizado:

```
getEtherPrice: async() => {
  try {
    const response = await axios.get('https://min-
api.cryptocompare.com/data/pricemulti?fsyms=ETH&tsyms=EUR&api_key=b52b375c592aed093dc49d82534
5687e7d04cbfc9b3737b30138f33f3acb092a');

    const ETH = Object.values(response.data);
    return ETH[0]["EUR"];
  } catch (err) {
    console.log(err);
  }
}
```

Figura 54: Lectura de datos remotos a través de Axios
Fuente: Elaboración propia

Por otro lado, necesitamos consultar el registro de la propiedad. Tras una investigación, se vio que este servicio es privado del estado, de pago y sólo es posible consultar información de propiedades propias, por lo cual, nos imposibilita saber qué usuario es propietario de qué vivienda.

Por este motivo, en esta prueba de concepto se ha optado por la creación de una API que simule al registro de la propiedad, la cual la DApp pueda consultar y así verificar que los usuarios son, efectivamente, los propietarios de un inmueble antes de publicarlo, y posteriormente, cambiar el propietario si este inmueble ha sido traspasado.

Ya que este punto ha supuesto un contratiempo, y el objetivo del proyecto no es crear una API, ya que la estructura del proyecto ya está construida, se ha hecho uso de Strapi [44], un framework de Node.js que nos provee de un CMS (sistema de gestión de contenidos) con un front-end integrado, capaz de crear colecciones para que después sean consumidas a través de una API.

En nuestro caso, hemos creado una colección llamada “Property”, dónde hemos incluido los siguientes datos:

The screenshot shows the Strapi Content-Type Builder interface. On the left sidebar, under 'Content-Type Builder', the 'Property' collection type is selected. The main panel displays the 'Property' collection type configuration with the following fields:

NAME	TYPE	OPTIONS
Idowner	Text	
Owner	Text	
Address	Text	
City	Text	
Zipcode	Text	
Idproperty	Text	

At the bottom of the list, there is a blue button labeled '+ Add another field to this collection type'.

Figura 55: Interfaz de Strapi
Fuente: Elaboración propia

La llamada a la API creada, se realiza de la misma manera que se ha hecho para consultar el precio del Ether, mediante Axios, en este caso desde el componente Vue.js properties, comprobando que coinciden los datos introducidos desde el front-end con los de la API.

4.6 ESTRUCTURA

Llegados a este punto, vemos la estructura de directorios y ficheros que forman el proyecto completo:

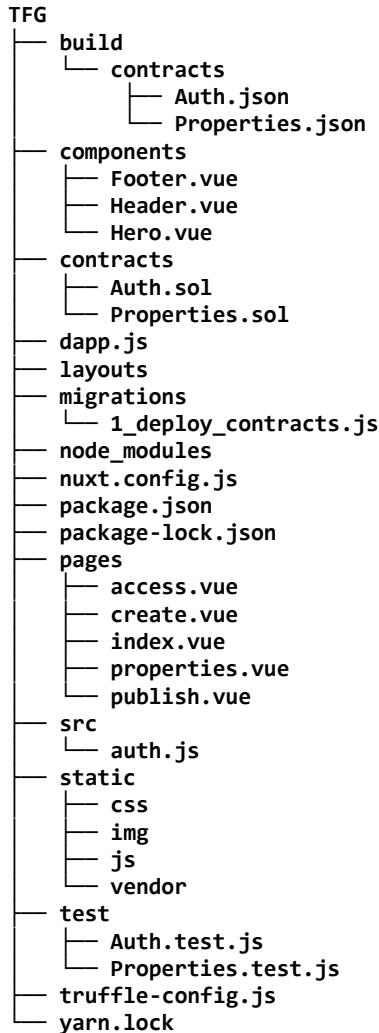


Figura 56: Estructura de ficheros del proyecto
Fuente: Elaboración propia

El directorio principal proyecto, al cual se le ha llamado TFG, está iniciado como una aplicación Nuxt, lo cual define una estructura predeterminada de directorios, como “components” y “pages”, los cuales incluyen componentes de Vue, además de otros directorios por defecto como “node_modules”, el cual contiene las dependencias NPM instaladas, o los ficheros “package.json”, los cuales contienen información del proyecto, así como sus dependencias en cuanto a módulos instalados.

El directorio “src”, creado manualmente, a posteriori, contiene el fichero “auth”, el cual se ocupa de gestionar la sesión de los usuarios y cuando se conectan o desconectan de la DApp.

Por su parte, el directorio “static”, contiene todo lo referente a elementos estáticos como imágenes o el código de estilos CSS global para todos los componentes.

Se han integrado también los directorios no orientados al código del front-end, dónde se incluyen el directorio “contracts”, que almacena los contratos inteligentes programados en Solidity, el directorio “migrations” que contiene el fichero de migraciones de Truffle, el directorio “test”, que incluye los tests de Truffle, así como el directorio “build”, el cual es esencial ya que almacena los ABI de los contratos inteligentes desplegados, y por tanto, son los ficheros con los que se comunica directamente la librería Web3.js.

También vemos el ya conocido fichero de configuración “truffle-config.js”, además del fichero “dapp.js”, el cual es el encargado de conectar front-end y blockchain.

En cuanto a la estructura de la API que simula al registro de la propiedad, es muy similar a la que crea inicialmente Nuxt, por lo que, no se entrará en su especificación, puesto que simplemente ha sido creada para ser consultada y modificada para satisfacer uno de los casos de uso.

4.7 PANTALLAS Y NAVEGACIÓN

Después de presentar las tecnologías y lenguajes utilizados que permitirán que la DApp cumpla con su propósito, en el presente apartado se muestran las pantallas, o interfaces de navegación, internas de la plataforma (Front-end), junto con las acciones disponibles en cada una de ellas:

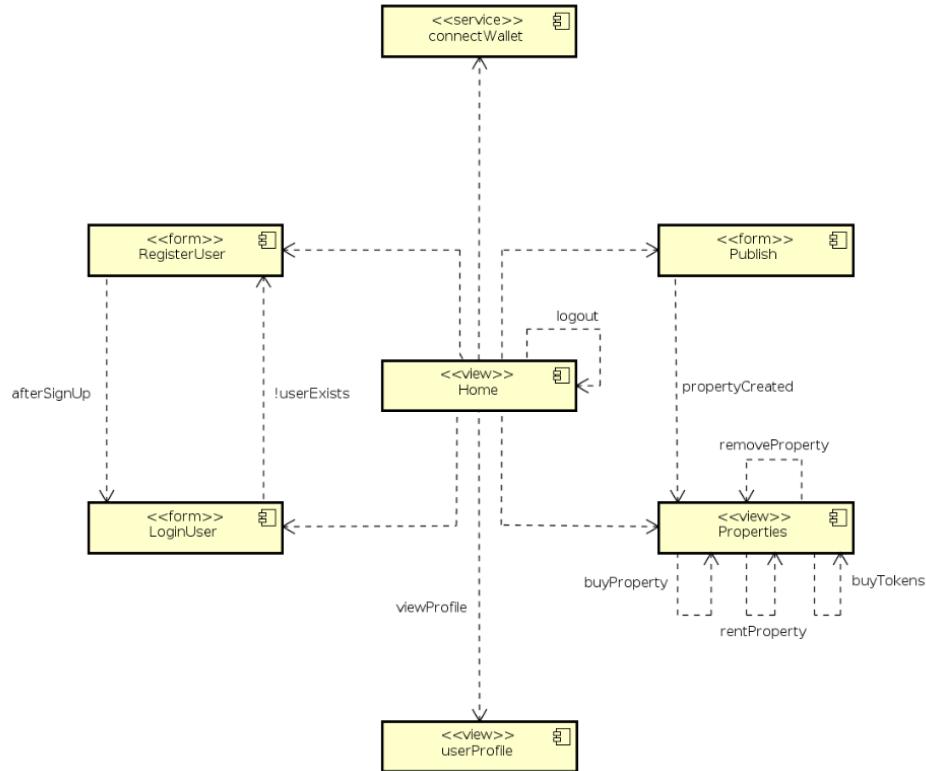


Figura 57: Diagrama de pantallas de navegación
Fuente: Elaboración propia

Para mantener el formato del documento, se mostrarán las diferentes pantallas de navegación en el Anexo 2.

5. CONCLUSIONES

5.1 CUMPLIMIENTO DE OBJETIVOS

El objetivo de este proyecto ha sido la exploración en el campo de la tecnología blockchain con tal de crear una prueba de concepto que pueda ayudar a mejorar el ámbito del sector inmobiliario en el futuro.

Durante el desarrollo del proyecto han surgido algunos contratiempos, cómo se podía esperar ya que es algo a tener en cuenta cuando se plantea un proyecto.

Sin embargo, podemos decir que, salvo la generación de NFTs para los usuarios que quisieran alquilar una propiedad por tokens, todos los objetivos han sido realizados. En algunos casos se ha tenido que cambiar el planteamiento inicial, pero llegando finalmente a obtener el mismo objetivo, cómo en la obtención del precio de la criptodivisa Ether actualizado, o en la obtención de datos procedentes del registro de la propiedad.

Personalmente considero que finalmente se cuenta con una plataforma web descentralizada que cumple con lo que se espera de ella, aportando a los usuarios que la utilicen una transparencia y veracidad en sus operaciones, al estar respaldadas por un smart contract y sin depender de un número desconocido de intermediarios, además de facilidad de uso, algo que no es lo habitual en este tipo de aplicaciones descentralizadas, ya que es un sector innovador que está comenzando, dónde la experiencia de usuario es todavía muy mejorable.

5.2 CONCLUSIONES PERSONALES

Inicialmente el planteamiento del proyecto fue complicado, debido a que, el hecho de trabajar de manera individual y no en grupo, hace que el proyecto se pueda plantear de manera subóptima, y que se tarde más en corregir o encarrilar un posible error de planificación. Pese a ello, estoy satisfecho con mi trabajo, considero que he organizado mi trabajo diario de una manera óptima, siguiendo una estructura de sprints semanales, mediante un KanBan de seguimiento que me ha permitido ir cumpliendo objetivos a lo largo del proyecto y mejorar mi eficiencia a la hora de organizarme.

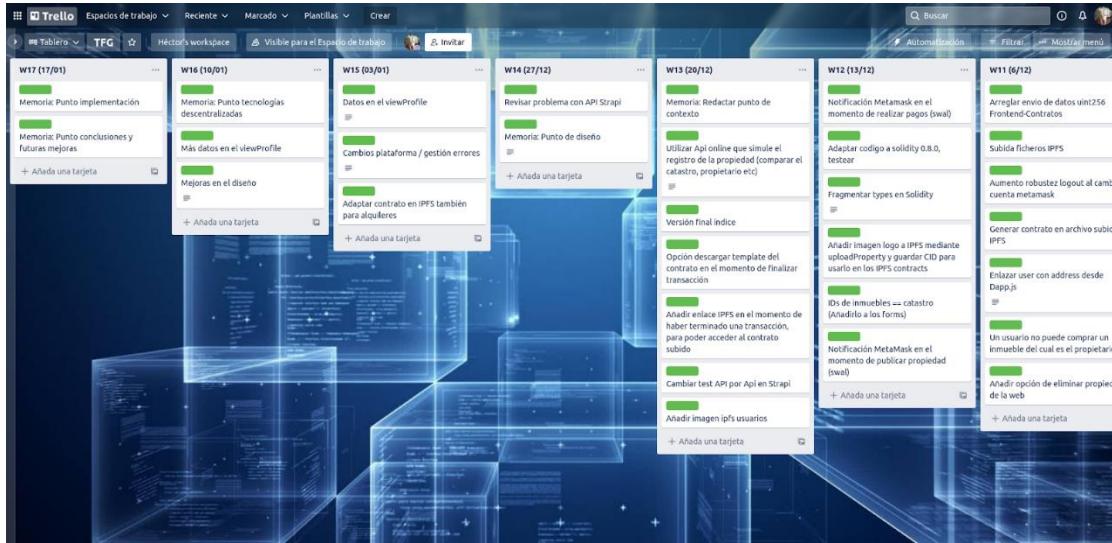


Figura 58: Estado final del KanBan personal
Fuente: Elaboración propia

Desde el punto de vista tecnológico, he aprendido muchos conceptos de la tecnología blockchain, que al inicio del proyecto creía, o sospechaba, que funcionaban de una manera, que posteriormente he visto que estaba equivocado. También he aprendido a programar en Solidity, así como el uso de las herramientas expuestas en el proyecto, además de mejorar mi nivel de programación con JavaScript y conocer algunos de sus frameworks, dónde todo el código ha quedado almacenado en un repositorio de GitHub propio [45].

5.3 FUTURAS MEJORAS

El proyecto en su estado actual, cumple con los objetivos fijados en el inicio, y sirve como prueba de concepto de una plataforma que, con algunos cambios podría existir y ser utilizable por usuarios reales.

Sin embargo, un punto interesante a implementar, sería poder aportar liquidez a los usuarios, de manera que puedan adquirir o alquilar propiedades, mediante transacciones periódicas. Actualmente, el hecho que impide llevar a cabo este punto, es la falta de liquidez por parte de la plataforma.

Si la plataforma estuviese respaldada, teniendo fondos suficientes, en el momento de ejecutar transacciones entre usuarios, la parte compradora, ejecutaría la compra o alquiler de un inmueble, y la plataforma abonará su totalidad al propietario. Posteriormente, la parte compradora podría ir abonando periódicamente, por ejemplo, mes a mes, la equivalencia del importe total a una dirección de clave pública donde el propietario sea la plataforma.

En este contexto, seguiría existiendo una descentralización, debido a que las condiciones estarían especificadas en un smart contract, cómo ahora, pero se les daría soporte a los usuarios a la hora de realizar transacciones de grandes importes, cómo sabemos que son los de los inmuebles.

Otro contratiempo para desplegar la plataforma y que fuese utilizada de manera real, es el hecho de consultar el registro de la propiedad, puesto que se tratan de datos privados y, pese que la tecnología blockchain cada vez está recibiendo más adopción por parte de organizaciones y gobiernos, este punto difícilmente cambiará y dichos datos seguirán siendo privados, por lo que, habría que aplicar otro tipo de autenticación de datos en un futuro, cómo por ejemplo, la subida de documentos acreditativos de la propiedad a la plataforma, de manera que no sean visibles, pero que permitan a los usuarios publicar propiedades de manera segura y verificada.

6. AGRADECIMIENTOS

A mi tutora Eva Marín, por aceptarme este tema para mi proyecto final de grado y por sus comentarios durante el seguimiento del proyecto.

A mi pareja, quién me ha visto trabajar, pegado al ordenador día y noche y siempre ha estado ahí para brindarme su apoyo incondicional.

A mi compañero y amigo Javier Delgado, por escucharme y darme ánimos, interesándose y dándome feedback del estado del proyecto en todo momento.

A los fundadores de mi empresa, Nectios, los cuales, además de mostrar su interés acerca del proyecto, han sido flexibles con mi horario durante estos meses, dándole prioridad a mi formación.

Por último, a mis amigos de toda la vida y a mi familia, por preocuparse por mí, durante el transcurso del proyecto.

7. BIBLIOGRAFÍA

- [0] Imran Bashir, Mastering Blockchain - Third Edition: Packt Publishing
- [1] Trello: <https://trello.com/es>
- [2] Git: <https://git-scm.com/about>
- [3] Metamask: <https://metamask.io/>
- [4] IPFS: <https://ipfs.io/#why>
- [5] How To Geek - API: <https://www.howtogeek.com/343877/what-is-an-api/>
- [6] Bitcoin: <https://bitcoin.org/bitcoin.pdf>
- [7] Hyperledger: <https://www.hyperledger.org/>
- [8] Alastria: <https://www.alastria.io/>
- [9] University of Virginia: https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf
- [10] Ethereum: <https://ethereum.org/es/>
- [11] Cardano: <https://cardano.org/>
- [12] Gwei: <https://gwei.io/index.html>
- [13] Solidity: <https://docs.soliditylang.org/en/v0.8.0/>
- [14] Web3university:
<https://web3university.webflow.io/getting-started#intro-to-blockchain>
- [15] Truffle: <https://trufflesuite.com/truffle/>
- [16] MDN - JavaScript: <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [17] Chaijs: <https://www.chaijs.com/>
- [18] Ganache: <https://trufflesuite.com/ganache/>
- [19] Chainlink: <https://chain.link/>
- [20] Infura: <https://infura.io/>
- [21] Ethereum: <https://ethereum.org/en/developers/docs/nodes-and-clients/>
- [22] Ycharts - Ethereum:
https://ycharts.com/indicators/ethereum_chain_full_sync_data_size
- [23] Go Ethereum Book - Faucets: <https://goethereumbook.org/en/faucets/>
- [24] Chainlink: <https://faucets.chain.link/kovan>

- [25] NPM - HDWalletProvider: <https://www.npmjs.com/package/@truffle/hdwallet-provider>
- [26] AnyBlockAnalytics: <https://www.anyblockanalytics.com/networks/ethereum/kovan/>
- [27] Chainlink - Data Feeds: <https://docs.chain.link/docs/ethereum-addresses>
- [28] Web3.utils: <https://web3js.readthedocs.io/en/v1.2.11/web3-utils.html>
- [29] Etherscan: <https://etherscan.io/>
- [30] GitHub - Truffle-suite: <https://github.com/trufflesuite/ganache-cli-archive>
- [31] Codecademy: <https://www.codecademy.com/resources/blog/what-is-a-framework/>
- [32] Vue.js: <https://es.vuejs.org/v2/guide>
- [33] MDN - HTML: <https://developer.mozilla.org/es/docs/Web/HTML>
- [34] MDN - CSS: <https://developer.mozilla.org/es/docs/Web/CSS>
- [35] Nuxt.js: <https://v3.nuxtjs.org/concepts/introduction>
- [36] Node.js: <https://nodejs.org/es/about>
- [37] Node Package Manager: <https://docs.npmjs.com/about-npm>
- [38] Web3.js: <https://web3js.readthedocs.io/en/v1.7.0/>
- [39] JSON-RPC: <https://www.jsonrpc.org/specification>
- [40] NPM - TruffleContract: <https://www.npmjs.com/package/truffle-contract>
- [41] IPFS: <https://docs.ipfs.io/reference/http/api/#api-v0-object-new>
- [42] CryptoCompare: <https://min-api.cryptocompare.com/>
- [43] Axios: <https://axios-http.com/docs/intro>
- [44] Strapi: <https://strapi.io/>
- [45] GitHub: <https://github.com/hmonpa/TFG>
- [46] MDN - Array:
https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/Array
- [47] Tutorialspoint - Mapping:
https://www.tutorialspoint.com/solidity/solidity_mappings.htm
- [48] MDN - SetInterval: <https://developer.mozilla.org/es/docs/Web/API/setInterval>

ANEXOS

ANEXO 1: SMART CONTRACTS

SMART CONTRACT AUTH

INTRODUCCIÓN Y ESTRUCTURAS DE DATOS

Los datos que maneja el contrato inteligente Auth, son de usuario, por lo que se ha declarado una estructura de datos del tipo struct, la cual define el tipo “User”:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Auth {

    uint public usersCounter = 0;
    struct User {
        address addr;           // Clave pública
        string name;
        string email;
        string password;
        string idCard;          // DNI
        string ipfsImage;        // Imagen de usuario
        bool isLoggedIn;
        uint256 createdAt;
    }

    mapping(address => User) public usersByAddr;
    User[] public users;

    // ...
}
```

Figura 59: Cabecera y estructuras de datos smart contract Auth
Fuente: Elaboración propia

Para cada usuario se definen:

- addr: Dirección de clave pública
- name: Nombre completo
- email: Correo electrónico
- password: Contraseña
- idCard: Documento identificativo del usuario (DNI)
- ipfsImage: Hash CID de la imagen de usuario
- isLoggedIn: Estado actual en la plataforma (Indica si está o no logueado)
- createdAt: Fecha y hora de creación en la plataforma

Cómo se puede ver en la “figura 59”, también se han declarado un array [46] llamado users, un mapping [47] llamado usersByAddr, y un contador, para llevar el control del número de usuarios existentes.

Lo siguiente que nos encontramos son eventos, concepto que no es demasiado común en los lenguajes de programación.

EVENTOS

```
event newUser(
    address addr,
    string name,
    string email,
    string password,
    string _idCard,
    string _ipfsImage,
    bool isLoggedIn,
    uint256 createdAt
);

event userLogged(
    address addr,
    string name,
    bool isLoggedIn
);

event userLogout(
    address addr,
    string name,
    bool isLoggedIn
);
```

Figura 60: Eventos smart contract Auth
Fuente: Elaboración propia

El hecho de invocar eventos en el interior de las funciones, permite que los argumentos pasados, queden almacenados en un registro de la transacción ejecutada. El evento se asocia a la dirección del contrato inteligente y se despliega en la blockchain.

FUNCIONES

En el presente apartado se mostrarán los headers de cada una de las funciones implementadas con una pequeña descripción:

- **signUp:**

```
function signUp(
    address _address,
    string memory _name,
    string memory _email,
    string memory _password,
    string memory _idCard,
    string memory _ipfsImage
) public returns (bool)
```

Figura 61: Función signUp smart contract Auth
Fuente: Elaboración propia

Registra un usuario a la plataforma, o lo que es lo mismo, crea un nuevo User y lo añade al mapping usersByAddr y al array users, devuelve un booleano en True, si no encuentra errores. En caso de registro satisfactorio, emite el evento newUser.

- **signIn:**

```
function signIn(
    address _address,
    string memory _password
) public returns (bool)
```

Figura 62: Función signIn smart contract Auth
Fuente: Elaboración propia

Comprueba que el par <Dirección Clave Pública, Contraseña> pasado por parámetro, sea idéntico al par guardado. En caso afirmativo, el campo isLoggedIn de User pasa a True, se emite el evento userLogged y se devuelve un booleano en True. En caso contrario, se devuelve un booleano en False.

- **getUser:**

```
function getUser(address _address) public view returns (address, string memory, string
memory, uint256, string memory, string memory)
```

Figura 63: Función getUser smart contract Auth
Fuente: Elaboración propia

Devuelve los datos de un usuario concreto dada su dirección de clave pública.

- **getAddrFromIndex:**

```
function getAddrFromIndex(uint _index) public view returns (address)
```

Figura 64: Función getAddrFromIndex smart contract Auth
Fuente: Elaboración propia

Devuelve una dirección de clave pública dado un índice del array users.

- **checkIfUserLoggedByAddr:**

```
function checkIfUserLoggedByAddr(address _address) public view returns (bool)
```

Figura 65: Función checkIfUserLoggedByAddr smart contract Auth
Fuente: Elaboración propia

Devuelve el estado de un usuario concreto, dada una dirección de clave pública, en la plataforma, en función de si está o no logueado.

- **logoutByAddr:**

```
function logoutByAddr(address _address) public
```

Figura 66: Función logoutByAddr smart contract Auth

Fuente: Elaboración propia

Cambia el campo isLoggedIn a False, y emite el evento userLogout.

Antes de pasar a especificar el contrato Properties, cabe comentar que todas las funciones son públicas, ya que serán visibles por todos los usuarios.

Además, en algunas funciones vemos especificado el modificador “view”, el cual le indica al compilador que en la función que lo incluya, no se va a almacenar ningún valor en el contrato, por lo tanto, no implicará ninguna transacción, lo que significa que la ejecución de dicha función no supondrá ningún tipo de gasto de comisiones de gas, a diferencia del resto de funciones, las cuales sí supondrán un gasto computacional para la red.

Existen otros modificadores que especificaremos más adelante cuando sean utilizados.

SMART CONTRACT PROPERTIES

INTRODUCCIÓN Y ESTRUCTURAS DE DATOS

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Properties {

    // Datos generales de una propiedad
    struct Property
    {
        string id;                                // Referencia catastral
        address owner;                            // Clave pública
        string city;
        string physicalAddr;                     // Dirección física
        uint256 price;
        uint256 rooms;
        uint256 area;
        uint256 bathrooms;
        uint sellOrRent;                         // Venta -> 1 : Alquiler -> 0
        uint256 createdAt;                       // Fecha de publicación
        uint256 soldOn;                           // Fecha de venta
    }

    // Imagen de una propiedad
    struct propertyImages
    {
        string idProperty;
        string ipfsImage;
    }

    // Propiedad en alquiler
    struct propertyForRenting
    {
        string idProperty;
        uint256 rentalEndDate;
    }

    // Propiedad en alquiler de manera tokenizada
    struct tokenizedProperty
    {
        string idProperty;
        uint256 rentalEndDate;
        uint256 tokens;
    }

    // Tokens de una propiedad
    struct propertyTokens
    {
        string idProperty;
        uint256 tokens;
    }

    // Tokens vendidos de una propiedad
    struct tokensPurchased
    {
        string idProperty;
        address owner;
        uint256 tokens;
    }

    // ...
}
```

Figura 67: Cabecera y estructuras de datos smart contract Properties - Parte I
Fuente: Elaboración propia

La estructura de datos principal, Property, contiene todas las propiedades que se publiquen en la plataforma, y para cada una de ellas se definen los siguientes campos:

- idProperty: Referencia catastral de la propiedad
- owner: Dirección de clave pública del usuario propietario
- city: Ciudad
- physicalAddr: Dirección física completa
- price: Precio
- rooms: Número de habitaciones
- area: Dimensiones en m²
- bathrooms: Número de baños
- sellOrRent: Tipo de propiedad (En venta o en alquiler)
- createdAt: Fecha y hora de publicación en la plataforma
- soldOn: Fecha y hora de venta (Hasta el momento de la venta es 0)

En el apartado en el que se presentó la Ethereum Virtual Machine, se indicó que esta trabaja con una pila virtual, o stack, por lo que Solidity limita el número de elementos que se pueden pasar por parámetro, para evitar que esta pila pueda llenarse. Esto hace que se haya tenido que declarar la estructura de datos propertyImages, para guardar las imágenes de las propiedades, la cual contiene los siguientes campos:

- idProperty: Referencia catastral de la propiedad
- ipfsImage: Hash CID de la imagen de la propiedad

La estructura propertyImages podrá relacionarse con Property mediante el campo idProperty.

Por otro lado, además de la limitación del número de argumentos que hemos comentado en el párrafo anterior, a diferencia smart contract Auth, en este caso se han declarado otras estructuras de datos adicionales mediante structs, para posteriormente, poder manipular correctamente la información que los usuarios modifican desde el Front-end.

Cómo se comentó en el apartado de especificación, las propiedades publicadas pueden tener como objetivo ser vendidas, alquiladas o alquiladas de manera tokenizada, por lo cual, era necesario hacer una diferenciación entre el tipo de propiedades declarando las siguientes estructuras:

- propertyForRenting: Define una propiedad publicada con el objetivo de ser alquilada, define los siguientes campos:
 - idProperty: Referencia catastral de la propiedad
 - rentalEndDate: Fecha fin de alquiler
- tokenizedProperty: Define una propiedad publicada con el objetivo de ser alquilada de manera tokenizada, define los siguientes campos:
 - idProperty: Referencia catastral de la propiedad
 - rentalEndDate: Fecha fin de alquiler
 - tokens: Número de tokens definidos en el momento de publicación
- propertyTokens: Define el número de tokens disponibles actualizado en cada momento, define los siguientes campos:
 - idProperty: Referencia catastral de la propiedad
 - tokens: Número de tokens disponibles, se actualiza después de cada compra

- tokensPurchased: Representa la compra de tokens de una propiedad por parte de un usuario, define los siguientes campos:
 - idProperty: Referencia catastral de la propiedad
 - owner: Dirección de clave pública del usuario comprador
 - tokens: Número de tokens adquiridos

Además, de la misma forma que se hizo en el smart contract Auth, se han definido diferentes mappings, contadores y arrays, para llevar un control general de las propiedades. En la siguiente figura se muestra cada estructura de datos con un comentario que indica su contenido o su propósito:

```
// properties: Contiene todas las propiedades
mapping (uint256 => Property) public properties;

// rentalProperties: Contiene las propiedades en alquiler
mapping (uint256 => propertyForRenting) public rentalProperties;

// tokenizedProperties: Contiene las propiedades en alquiler de manera tokenizada
mapping (uint256 => tokenizedProperty) public tokenizedProperties;

// propertyImages: Contiene las imágenes de las propiedades
mapping (uint256 => propertyImages) public propertyImg;

// startedTokens: Contiene los tokens iniciales de cada propiedad
mapping (uint256 => propertyTokens) public startedTokens;

// ownershipTokens: Contiene la titularidad (o propiedad) de los tokens de cada propiedad
mapping (uint256 => tokensPurchased) public ownershipTokens;

// props[]: Contiene todas las propiedades
Property[] public props;

// Contador del número de propiedades
uint public cnt = 0;

// Contador del número de tokens adquiridos por usuarios
uint public cntTokens = 0;
```

Figura 68: Estructuras de datos smart contract Properties - Parte II
Fuente: Elaboración propia

EVENTOS

Del mismo modo que se ha hecho en el smart contract Auth, se han definido eventos para que se puedan emitir dentro de las funciones cuando ocurran cambios en las estructuras de datos definidas:

```
// Nueva propiedad publicada
event PropertyCreated(
    string id,
    address owner,
    string city,
    string physicalAddr,
    uint256 price,
    uint sellOrRent,
    uint256 createdAt
);

// Nueva propiedad publicada en alquiler
event PropertyForRentingCreated(string idProperty, uint256 rentalEndDate);

// Nueva propiedad publicada en alquiler de manera tokenizada
event TokenizedPropertyCreated(string idProperty, uint256 rentalEndDate, uint256 tokens);

// Propiedad vendida
event propertySold (address soldBy, uint256 price, uint256 soldOn);

// Propiedad alquilada
event propertyRented (address rentedBy, uint256 price, uint256 rentalEndDate);

// Token(s) adquirido(s) en una propiedad alquilada de manera tokenizada
event propertyTokenPurchased (address purchasedBy, string idProperty, uint256 numberoftokens, uint256 pricePerToken);

// Propiedad eliminada
event propertyRemoved (address byOwner, string id);
```

Figura 69: Eventos smart contract Properties
Fuente: Elaboración propia

FUNCIONES

- **uploadProperty:**

```
function uploadProperty(string memory _id, address _owner, string memory _city, string memory
_physicalAddr, uint256 _price, uint256 _sellOrRent, uint256 _tokens, uint256 _rentalEndDate,
string memory _ipfsImage) public
```

Figura 70: Función uploadProperty smart contract Properties
Fuente: Elaboración propia

Publica una propiedad en la plataforma, o lo que es lo mismo, crea una nueva Property y la añade al mapping properties y al array props. Emite el evento PropertyCreated, y en caso de tratarse de una propiedad en alquiler, o una propiedad en alquiler de manera tokenizada, emite los eventos PropertyForRentingCreated o TokenizedPropertyCreated, respectivamente. Además, en caso de tratarse de una propiedad en alquiler, o una propiedad en alquiler de manera tokenizada, la añade al mapping rentalProperties o a los mappings tokenizedProperties y startedTokens, respectivamente.

- **addPropertyData:**

```
function addPropertyData(uint256 _numRooms, uint256 _area, uint256 _bathrooms) public
```

Figura 71: Función addPropertyData smart contract Properties
Fuente: Elaboración propia

Da soporte a la función uploadProperty, la cual registra un gran número de argumentos, por lo tanto, algunos argumentos son introducidos desde esta función auxiliar en el momento de publicar una propiedad, lo cual es totalmente transparente para el usuario.

- **sendBalance:**

```
function sendBalance(address payable _receiver, uint256 _amount) payable external
```

Figura 72: Función sendBalance smart contract Properties
Fuente: Elaboración propia

Permite enviar pagos a una dirección de clave pública determinada, es una función tipo “external”, es decir, sólo es accesible cuando se la llama desde fuera del contrato inteligente. También incluye el modificador “payable”, el cual permite enviar y recibir Wei.

- **buyProperty:**

```
function buyProperty(address payable _address, string memory _id) public payable
```

Figura 73: Función buyProperty smart contract Properties
Fuente: Elaboración propia

Permite realizar la acción de comprar una propiedad determinada. Llama a la función sendBalance pasándole como parámetros la dirección de clave pública del vendedor y la variable global de Solidity “msg.value”, la cual permite enviar un número de Wei determinado en una función.

Si la transacción se realiza correctamente, se emite el evento propertySold y la dirección de clave pública del propietario para ese objeto Property, es modificada en el mapping properties y en el array prop.

- **rentProperty:**

```
function rentProperty(address _address, string memory _id, uint256 _rentalEndDate) public payable
```

Figura 74: Función rentProperty smart contract Properties
Fuente: Elaboración propia

Permite realizar la acción de alquilar una propiedad determinada. Similarmente a la función buyProperty, llama a la función sendBalance para realizar el pago de la transacción al propietario, y en caso de realizarse satisfactoriamente, se emite el evento propertyRented, en este caso el propietario no se modifica al tratarse de un alquiler.

- **buyTokens:**

```
function buyTokens(address _from, uint256 _numTokens, string memory _id) public payable
```

Figura 75: Función buyTokens smart contract Properties
Fuente: Elaboración propia

Permite realizar la acción de adquirir tokens de una propiedad determinada. Similarmente a las funciones anteriores, llama a la función sendBalance para realizar el pago de la transacción al propietario, y en caso de realizarse satisfactoriamente, se emite el evento propertyTokenPurchased. Además, modifica el mapping ownershipTokens, añadiendo los tokens adquiridos por un usuario, modifica el número de tokens de la propiedad en cuestión y aumenta el contador del número de tokens que han sido adquiridos.

- **removeProperty:**

```
function removeProperty(string memory _id) public
```

Figura 76: Función removeProperty smart contract Properties
Fuente: Elaboración propia

Elimina una propiedad publicada anteriormente, que no había sido vendida. Utiliza la opción “delete” de Solidity para eliminar la propiedad del mapping properties y del array props, emite el evento propertyRemoved.

Cabe destacar que, aunque se elimine la propiedad de una estructura de datos, en el histórico de la red siempre estará presente, siguiendo el principio de inmutabilidad, no obstante, esta función ha sido programada para que, si un usuario comete un error a la hora de publicar una propiedad, pueda revertirlo.

- **propertySettled:**

```
function propertySettled(uint256 _index) public
```

Figura 77: Función propertySettled smart contract Properties

Fuente: Elaboración propia

Actualiza el campo soldOn en una propiedad concreta, añadiendo una fecha y hora de venta en base al timestamp del bloque actual de la red, y la actualiza en el mapping properties y en el array props.

- **getPropertyById:**

```
function getPropertyById(string memory _id) public view returns (uint)
```

Figura 78: Función getPropertyById smart contract Properties

Fuente: Elaboración propia

Dado el identificador de una propiedad, devuelve el índice en el que se encuentra en el mapping properties y en el array props, en caso de no encontrarlo devuelve una excepción.

- **getPropertyOwner:**

```
function getPropertyOwner(string memory _id) public view returns (address)
```

Figura 79: Función getPropertyOwner smart contract Properties

Fuente: Elaboración propia

Dado un identificador de una propiedad, devuelve la dirección de clave pública de su propietario.

ANEXO 2: PANTALLAS DE NAVEGACIÓN

En el presente anexo se mostrarán cada una de las pantallas del front-end desarrollado, junto con su propósito y la navegación que hay entre ellas.

- **Home:**

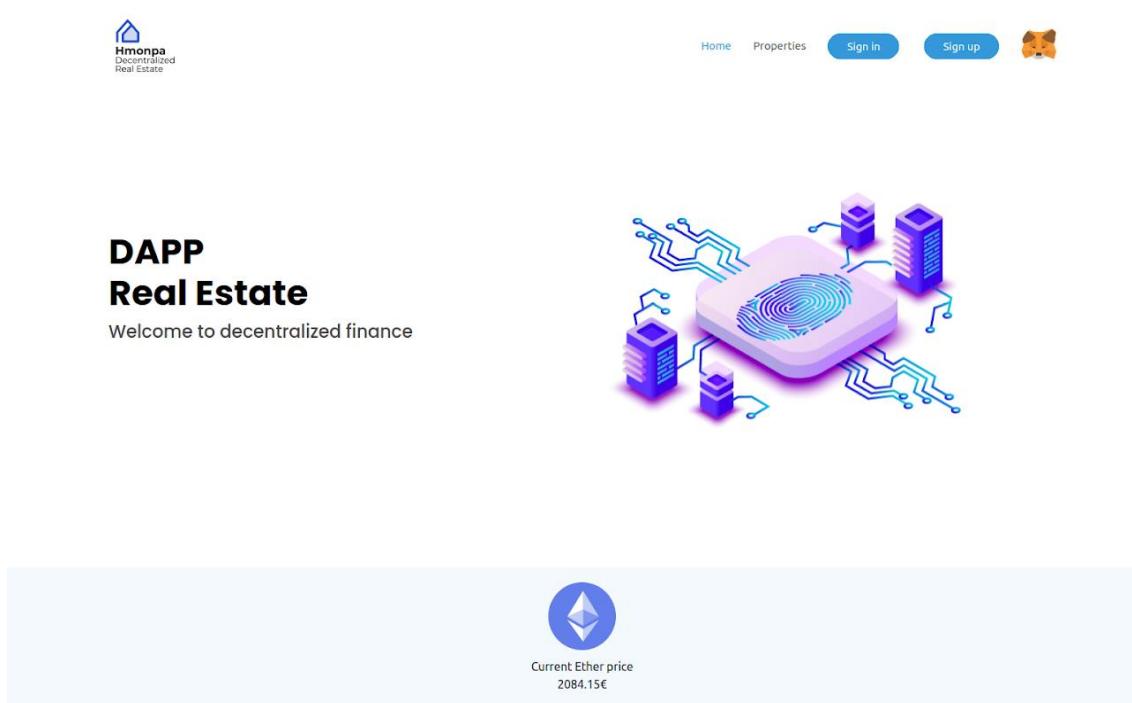


Figura 80: Pantalla de inicio

Corresponde al componente Hero.vue. Es la pantalla inicial que los usuarios ven al acceder a la plataforma, muestra el precio del Ether, el cual se actualiza de manera reactiva cada segundo, gracias a la función setInterval [48] nativa de JavaScript, la cual permite que, en este caso, la función getEtherPrice (figura 54), se repita con un tiempo de retraso entre cada ejecución, en este caso cada segundo.

Todas las pantallas incluyen el header (componente Header.vue), el cual corresponde al encabezado de la página, contiene un botón para conectar una billetera Metamask, así como enlaces a las pantallas "Home", "Properties", "Sign in" y "Sign up", cuando el usuario no esté logueado. En caso de estar logueado, contendrá enlaces a "Home", "Properties", "Publish" y "Logout".

- **Sign up:**

The screenshot shows the 'SIGN UP' page of the Hmonpa DApp. At the top, there's a logo for 'Hmonpa Decentralized Real Estate'. Below the logo, there are navigation links for 'Home', 'Properties', 'Sign in' (highlighted in blue), 'Sign up' (highlighted in blue), and a user icon. The main form has five input fields: 'Full name' (Héctor Montesinos Parra), 'ID Card' (01234567P), 'Email' (example@testing.eth), 'Password' (*****), and 'Upload image' (button to select file). At the bottom is a 'Register' button.

Figura 81: Pantalla de registro de usuarios

Corresponde al componente `create.vue`. Los usuarios que quieran crear un usuario en la plataforma, deberán conectar una billetera en Metamask, ya que será el identificador de usuario, de lo contrario, sólo podrán visualizar las pantallas “Home”, “Properties”, “Sign in” y “Sign up”, obviamente sin poder adquirir propiedades, registrarse ni loguearse.

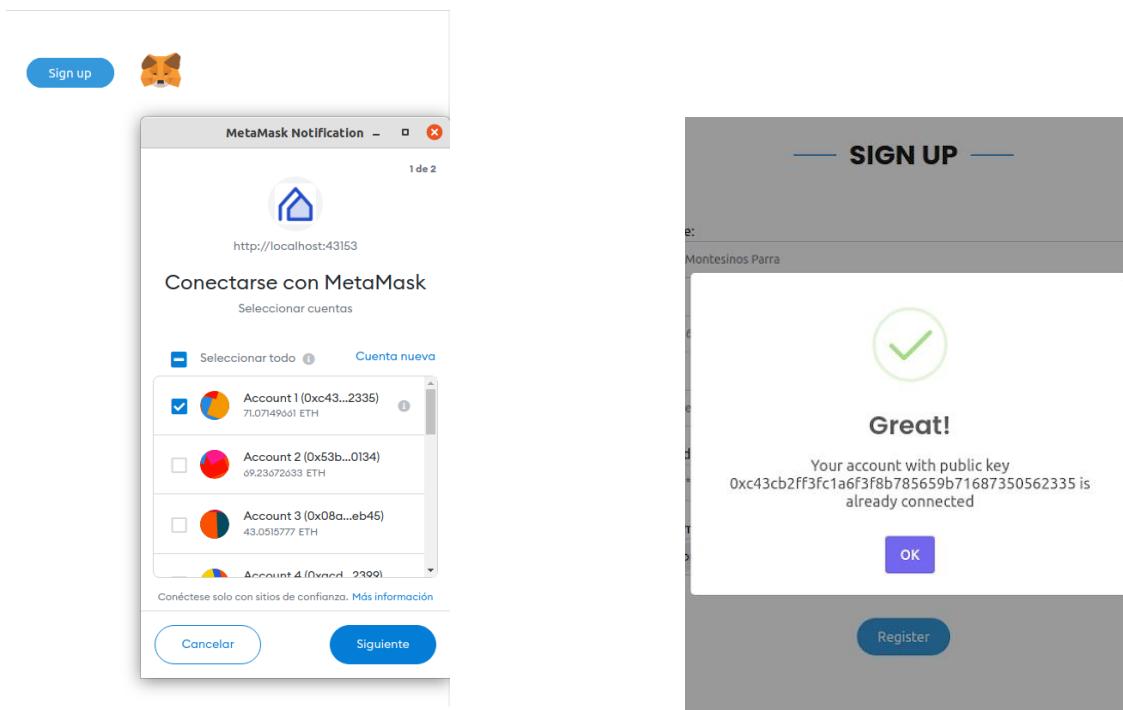


Figura 82: Proceso de conexión con la billetera Metamask

Figura 83: Modal de conexión con la billetera Metamask

- **Sign in:**

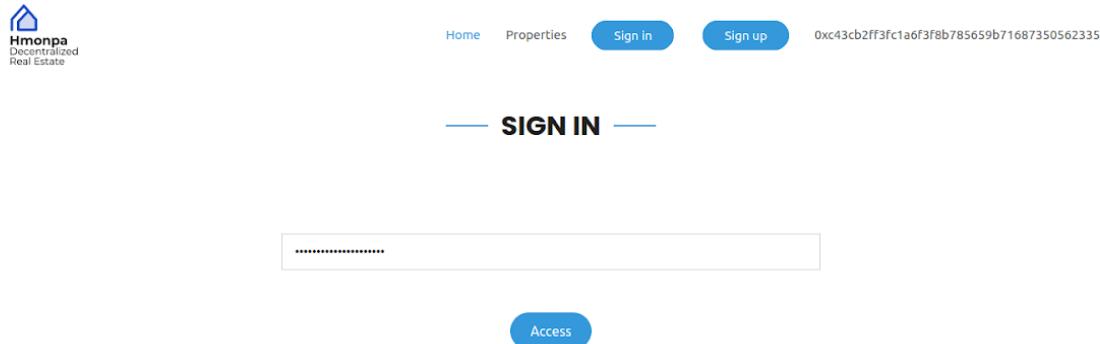


Figura 84: Pantalla de login

Corresponde al componente access.vue. Cuando los usuarios conectan su billetera Metamask a la plataforma, el botón para conectar la billetera desaparece y ya pueden registrarse. Una vez registrados, se les redirige a la pantalla de login, dónde el identificador de usuario será la clave pública de la billetera conectada, visible en todo momento en la parte superior derecha. Si introducen erróneamente la contraseña, aparece un modal de error, en caso contrario se les redirige a la pantalla de inicio.

Si la conexión resulta satisfactoria, desaparecerán los botones de registro y login, apareciendo así, un botón de logout y una sección dónde los usuarios podrán consultar sus datos personales:

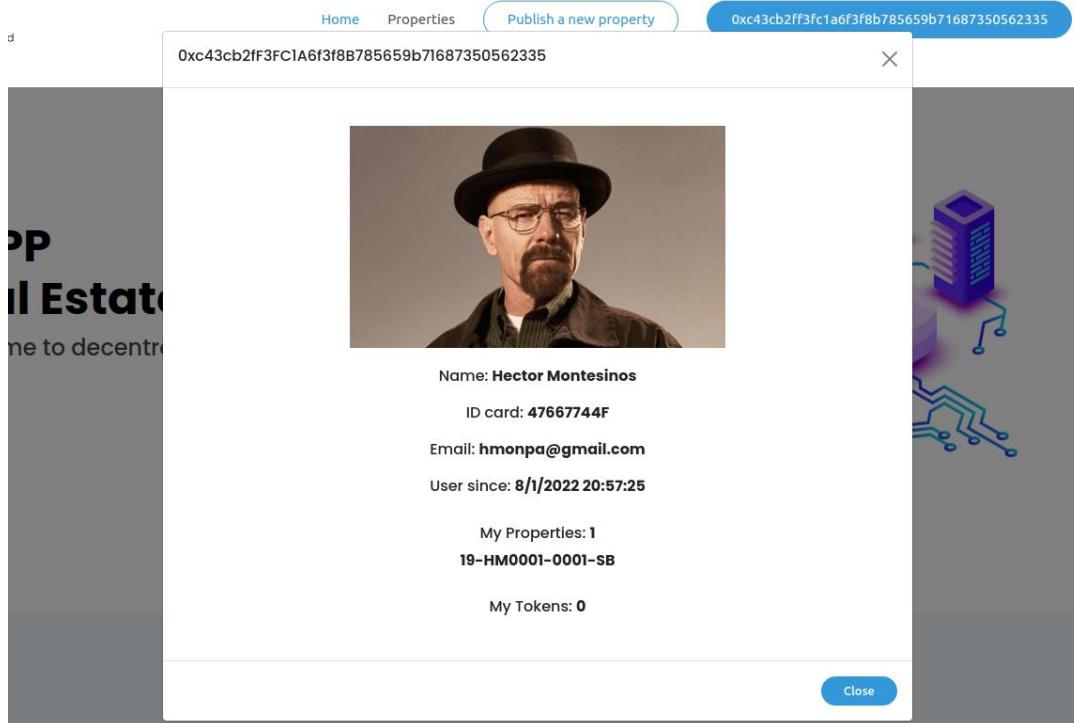


Figura 85: Modal detalles de usuario

- **Publish:**

The screenshot shows the 'UPLOAD PROPERTIES' form. At the top, there are navigation links: Home, Properties, Publish a new property (highlighted in blue), a unique identifier (0xc43cb2ff3fc1a6f3f8b785659b71687350562335), and Logout. The main form fields include:

- Cadastral reference: XX-XXXXX-XXXX-XX
- City: Barcelona
- Address: Plaça Sant Jaume, 1
- Price in EUR: 150000
- Rooms: A slider set to 1.
- Area in m²: 80
- Bathrooms: A slider set to 1.

Below these fields are two buttons: 'Sell' (blue) and 'Rent' (white). Further down is an 'Upload image:' section with a 'Seleccionar archivo' button and a message 'Ningún archivo seleccionado'. A large blue 'Publish' button is at the bottom right of the form area.

Figura 86: Pantalla publicación de propiedades - Parte I

Corresponde al componente publish.vue. Los usuarios pueden publicar inmuebles introduciendo todos los datos con los que se almacenarán posteriormente en el smart contract Properties. Por defecto, la opción que aparece en pantalla inicialmente, tiene como objetivo poner en venta el inmueble, pero los usuarios pueden hacer click en la opción “Rent”, para publicar inmuebles en alquiler:

The screenshot shows the 'Publish a new property' interface. At the top, there are navigation links for 'Home', 'Properties', and a blue button 'Publish a new property'. To the right is a wallet address: '0xc43cb2ff3fc1a6f3f8b785659b71687350562335' and a 'Logout' link. The main form area contains fields for 'Rooms' (set to 1), 'Area in m²' (set to 80), and 'Bathrooms' (set to 1). Below these are two buttons: 'Sell' (white background) and 'Rent' (blue background). Further down are fields for 'Rental end date' (set to 08/08/2022) and 'Upload image' (button labeled 'Seleccionar archivo' and placeholder 'Ningún archivo seleccionado'). A large blue 'Publish' button is at the bottom.

Figura 87: Pantalla publicación de propiedades - Parte II

Si el inmueble tiene una única habitación, como ocurre en la “figura 87”, al usuario no se le mostrará la opción de publicar un inmueble en alquiler por tokens. Sin embargo, si el número de habitaciones es superior, dicha opción dejará de estar oculta:

This screenshot shows the same publishing form as Figure 87, but for a property with two rooms. The 'Rooms' field is set to 2. A new section titled 'Is it tokenized?' is present, featuring a blue toggle switch that is turned on. The rest of the form fields and layout are identical to Figure 87, including the 'Sell' and 'Rent' buttons, rental end date, image upload, and a large blue 'Publish' button at the bottom.

Figura 88: Pantalla publicación de propiedades - Parte III

En cuanto a restricciones, si un usuario intenta publicar una propiedad al consultar sus datos contra el registro de la propiedad, se verifica que no es el titular de la propiedad, o si la propiedad ya está publicada en la plataforma, aparecen modales de restricción:

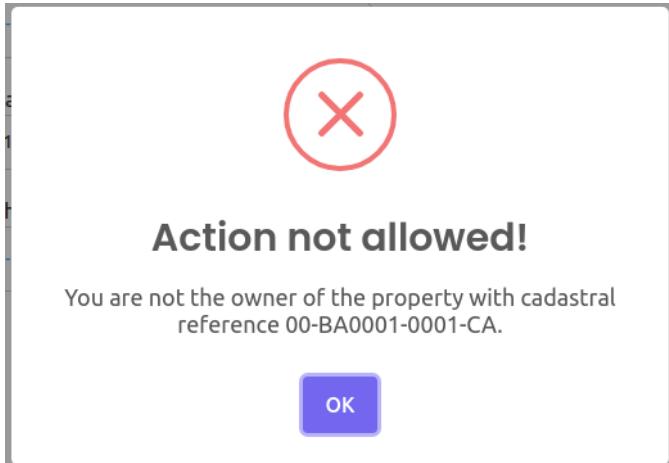


Figura 89: Modal de restricción por invalidez de datos, en pantalla Publish

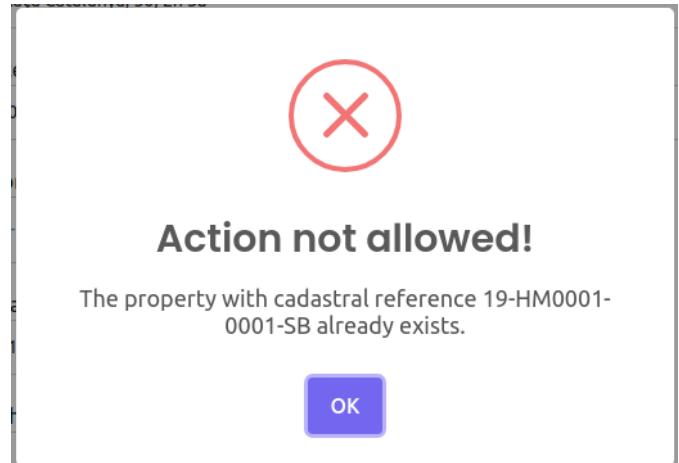


Figura 90: Modal de restricción por propiedad existente, en pantalla Publish

Si se comprueba que la propiedad no existe en la plataforma, y se verifica que el usuario es el titular de la propiedad, se permite su publicación, dónde, al usuario el aparece un aviso, y si está seguro de querer publicar su propiedad en la plataforma, podrá aceptar haciendo clic, y le aparecerá un mensaje de confirmación:

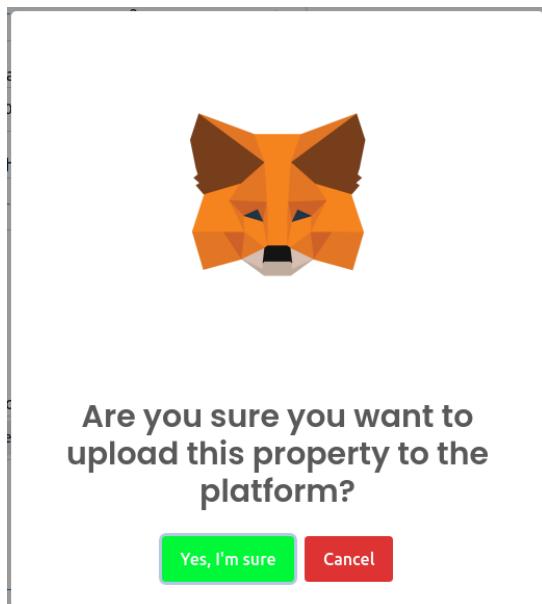


Figura 91: Modal de aprobación de publicación, en pantalla Publish

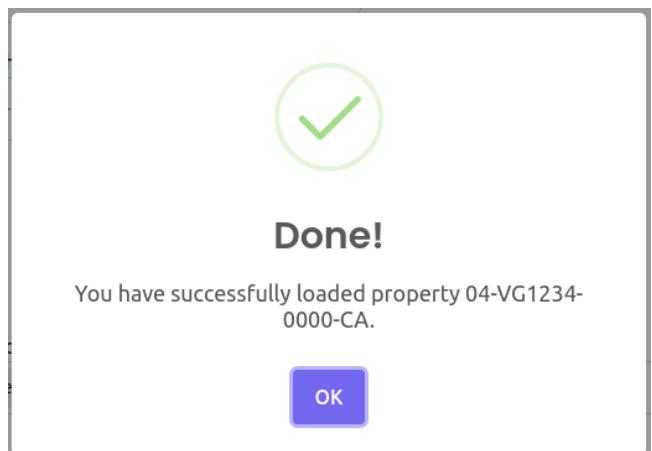


Figura 92: Modal de confirmación de publicación, en pantalla Publish

- **Properties:**

The screenshot shows the 'PROPERTIES' section of the Hmonpa Real Estate platform. At the top, there are navigation links for 'Home', 'Properties', 'Publish a new property', a user ID '0xc43cb2ff3fc1a6f3f8b785659b71687350562335', and 'Logout'. Below the header, a banner says 'Check out the current properties uploaded to the platform!' with a dropdown menu set to 'All' and a search bar. The main area displays four property cards in a grid:

Property Type	Location	Price (ETH)	Details
RENT	Cunit	90000€ (40.27 ETH)	Posted on 8/1/2022 20:57:23
SALE	Sant Boi de Llobregat	120000€ (53.74 ETH)	Posted on 8/1/2022 20:57:23
SOLD	Viladecans		Rented on 8/1/2022 20:58:32
SALE	Vilanova i la Geltrú	150000.00€ (67.12 ETH)	Posted on 23/1/2022 11:03:46

Figura 93: Pantalla de propiedades

Corresponde al componente properties.vue. En esta pantalla, los usuarios pueden consultar las propiedades publicadas por otros usuarios, o por ellos mismos. En ella pueden filtrar si quieren ver sólo sus propiedades, o todas, incluso filtrar por ciudad.

Hay diferenciación entre las propiedades en venta, en alquiler o las que ya han sido alquiladas o vendidas.

Las propiedades en alquiler por tokens, aparecen como en alquiler, y las propiedades que ya han sido vendidas o alquiladas, aparecen marcadas como tal, mediante un icono como se aprecia en la "figura 93".

Al hacer clic sobre una propiedad, aparece un modal con su información y un botón que posibilita la opción de comprarla o alquilarla, en caso de que el usuario no sea el titular de la misma, en cuyo caso dicho botón no aparecerá:

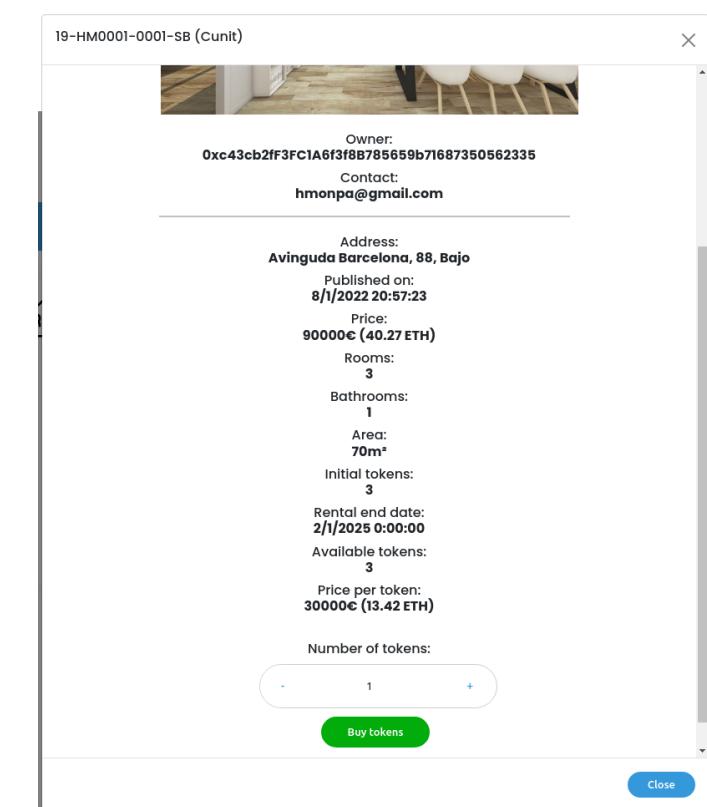
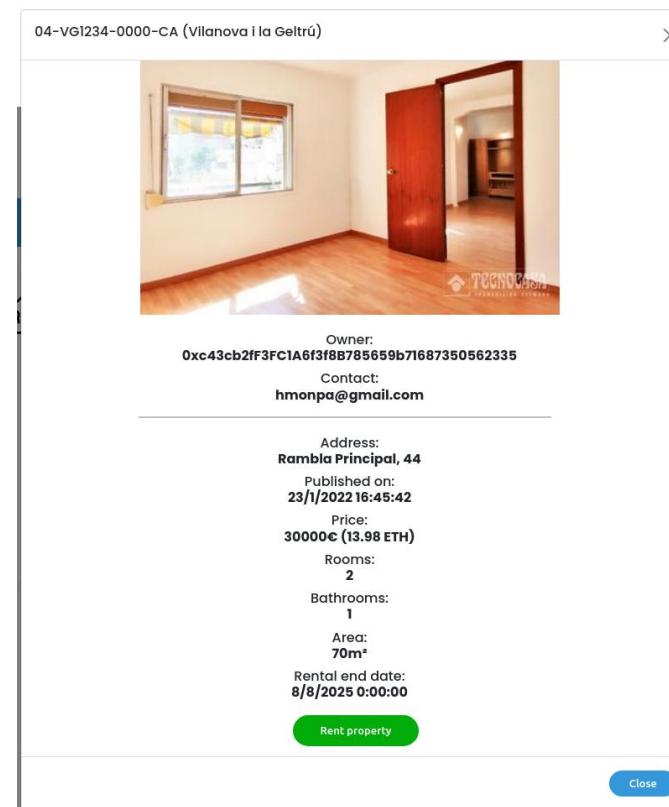
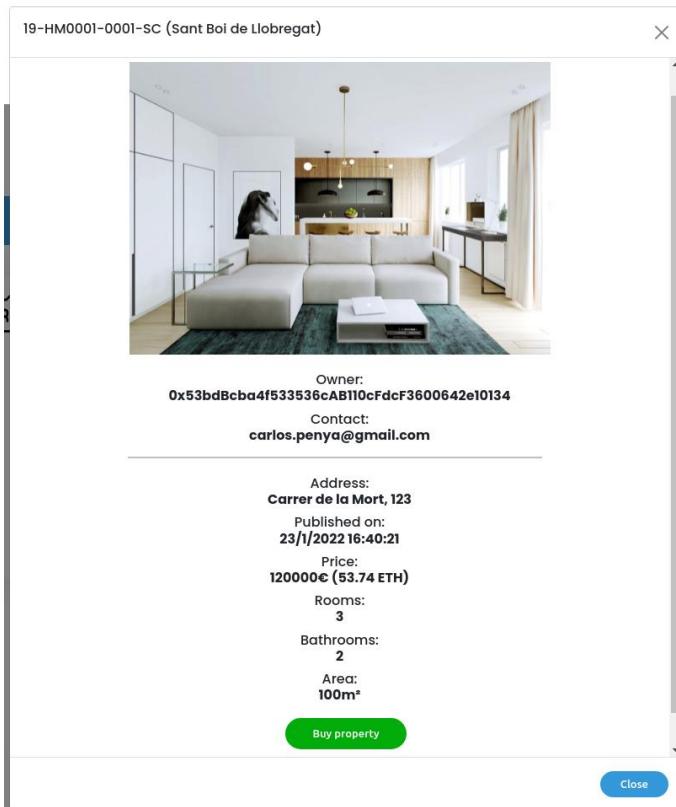


Figura 94: Modal de propiedad en venta

Figura 95: Modal de propiedad en alquiler

Figura 96: Modal de propiedad en alquiler por tokens

Si el usuario es el titular de una propiedad publicada, y quiere hacer cambios en el precio o simplemente, finalmente no quiere venderla o alquilarla, puede retirarla de la plataforma:

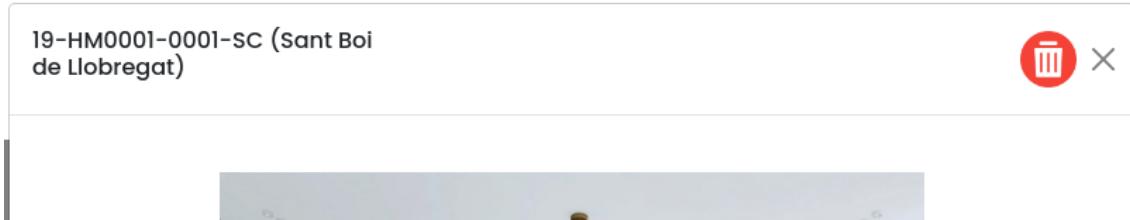


Figura 97: Opción de eliminar propiedad desde modal

Cuando los usuarios quieren adquirir o alquilar propiedades, o adquirir tokens, mediante los botones de las figuras 94, 95 y 96, respectivamente, les aparece un modal para que aprueben la transacción:

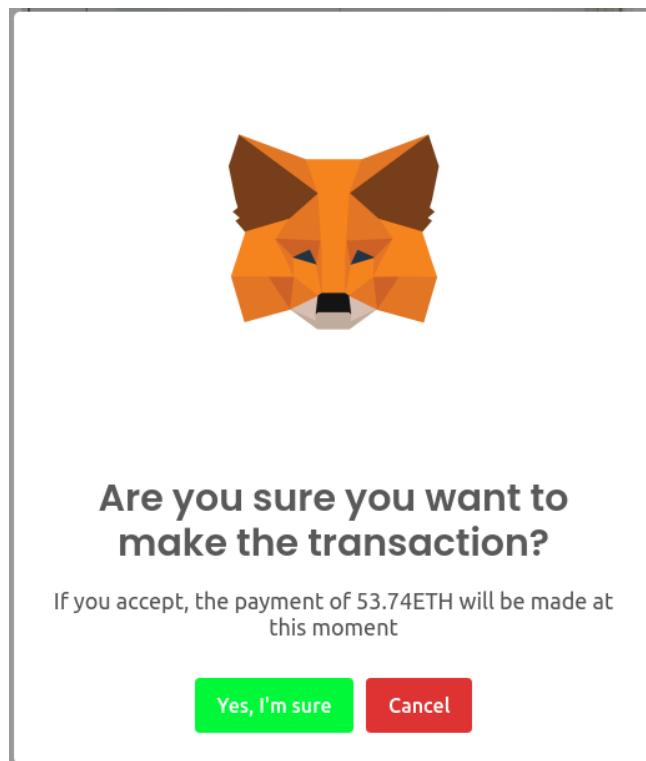


Figura 98: Modal de aprobación de transacción

Si aprueba la realización de la transacción, aparece un modal de confirmación, conforme han transferido los fondos en Ether al, hasta ese mismo momento, titular de la propiedad:

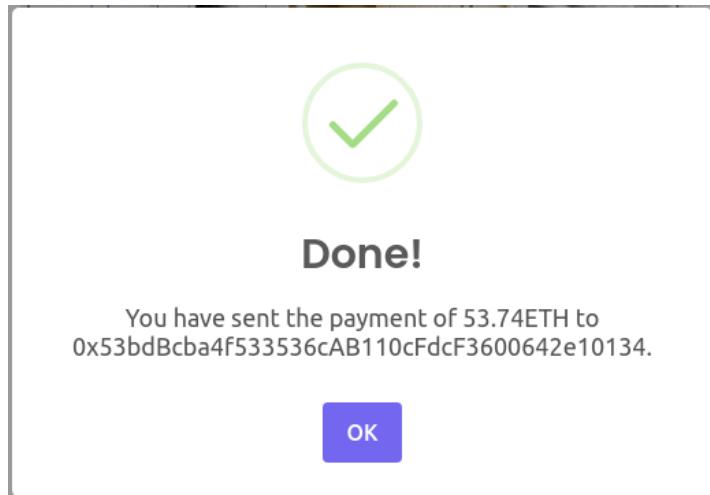


Figura 99: Modal de confirmación de transacción

Posteriormente, al usuario se le muestra el hash CID del contrato de compraventa o alquiler, el cual ha sido subido a IPFS automáticamente, para que pueda guardarlo y acceder siempre a él, además de un enlace directo al mismo. También tiene la opción de descargar el contrato en formato PDF en su propio dispositivo:

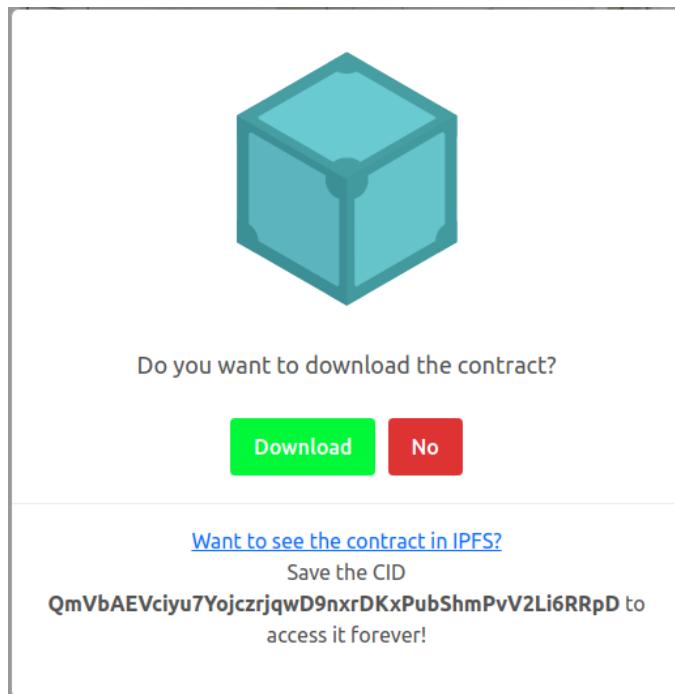


Figura 100: Modal con información del contrato generado

Haciendo clic en el enlace de la “figura 99”, o haciendo clic en “Download”, en esa misma figura, el usuario puede ver el contenido de su contrato de compraventa o alquiler:



Figura 101: Visualización de contrato generado en IPFS

ANEXO 3: DIAGRAMAS DE GANTT

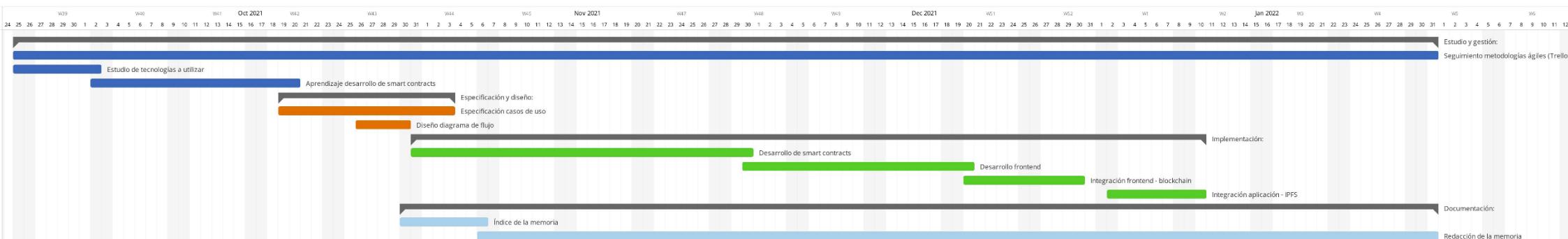


Figura 102: Diagrama de Gantt inicial
Fuente: Elaboración propia



Figura 103: Diagrama de Gantt final
Fuente: Elaboración propia