



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

Second Hand Chain



Presentado por Áxel Rubio González
en Universidad de Burgos — 3 de julio de 2023
Tutor: Jesús Manuel Maudes Raedo
Co-tutor: Sandra Rodríguez Arribas
Tutor de Empresa: Julia Zuara Jiménez



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Jesús Manuel Maudes Raedo y D. Sandra Rodríguez Arribas, profesores del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Áxel Rubio González, con DNI 13173823K, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Second Hand Chain.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de julio de 2023

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

D. Jesús Manuel Maudes Raedo

D. Sandra Rodríguez Arribas

Resumen

Actualmente el mercado de compra venta de teléfonos móviles esta experimentando un gran ascenso a nivel global. Dicho ascenso acarrea nuevas preocupaciones para los compradores que pueden desconfiar del uso previo de dichos teléfonos móviles, ocasionando pérdidas económicas a los diferentes actores involucrados en el mercado.

Para mitigar estas pérdidas debidas a la desconfianza de los diferentes usuarios, *Second Hand Chain* propone el empleo de la tecnología *Blockchain* para poder verificar acciones de compraventa entre particulares empleando como relación entre teléfono y su registro en *Blockchain* el IMEI. El principal objetivo de este proyecto es que los usuarios puedan interactuar directamente sin la dependencia de terceros para la custodia de los datos de sus teléfonos.

Para ello, en este trabajo se implementa una aplicación web completamente funcional que interactúa con un *smart contract* desplegado en la *testnet Sepolia* de *Ethereum*, permitiendo a los usuarios gestionar esta compraventa de teléfonos, generando valor a todos los actores en este mercado.

Descriptores

Blockchain, Ethereum, Sepolia, React, Dapp, proveedor de nodos, NFT, erc721, testnet, metadatos de NFT

Abstract

Currently, the mobile market for buying and selling telephones is experiencing a great rise globally. Said rise brings new concerns for buyers who may mistrust the previous use of such mobile phones, causing economic losses to the different players involved in the market.

To reduce these losses due to the mistrust of different users, *Second Hand Chain* proposes the use of *Blockchain* technology to be able to verify purchases between individuals using the IMEI as a relationship between the phone and its record on the *Blockchain*. The main objective of this project is that users can interact directly without the dependence on third parties for the custody of the data about their phones.

For this purpose, in this work, a fully functional web application is implemented that interacts with an intelligent contract deployed in the *Sepolia Ethereum testnet*, allowing users to manage this sale of telephones, generating value for all the actors in this market.

Keywords

Blockchain, Ethereum, Sepolia, React, Dapp, node provider, NFT, ERC721, testnet, NFT metadata

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vi
Introducción	1
1.1. Estructura de la memoria	2
Objetivos del proyecto	3
2.1. Objetivos teóricos	3
2.2. Objetivos técnicos	4
2.3. Objetivos personales	4
2.4. Entregables	5
Conceptos teóricos	7
3.1. Introducción a Blockchain	7
3.2. Funcionamiento de Blockchain	8
3.3. <i>Ethereum</i>	14
3.4. <i>IPFS</i> y <i>Filecoin</i>	18
Técnicas y herramientas	21
4.1. Metodologías de gestión de proyectos	21
4.2. Patrones de diseño	22
4.3. Herramientas para el desarrollo del <i>backend</i>	26
4.4. Herramientas para el desarrollo del <i>frontend</i>	28
4.5. Control de Versiones	34

4.6. Otras herramientas utilizadas	35
Aspectos relevantes del desarrollo del proyecto	37
5.1. Fases iniciales	37
5.2. Desarrollo <i>backend</i>	41
5.3. Desarrollo <i>Frontend</i>	45
5.4. Despliegue	46
Trabajos relacionados	49
Conclusiones y Líneas de trabajo futuras	51
7.1. Conclusiones	51
7.2. Líneas de trabajo futuras	53
Bibliografía	55

Índice de figuras

3.1. Merkle Tree[40]	9
3.2. Ejemplo de clave pública y clave privada en una de las <i>wallet</i> empleadas para crear Second Hand Chain	10
3.3. Ejemplo de bloques en una <i>Blockchain</i> . Extraído de [11]	11
3.4. Diagrama de funcionamiento de las <i>gas fees</i> para una transacción. Extraído de: [14]	17
3.5. Costes de <i>gas fees</i> para un bloque real de <i>Ethereum Sepolia</i> . Extraído de [17].	18
4.1. <i>Struct</i> que representa un teléfono en Second Hand Chain	23
4.2. <i>memory array building</i> en Second Hand Chain	25
4.3. Ejemplo de uso de tipos en Second Hand Chain	29
4.4. Ejemplo código en <i>HTML</i> . Extraído de [21]	30
4.5. Ejemplo código en <i>JSX</i> . Extraído de [21]	31
4.6. Ejemplo uso conjunto de <i>Tailwind css</i> y <i>DaisyUi</i>	33
5.1. Diagrama de secuencias inicial.	39
5.2. Diagrama de casos de uso inicial.	40
5.3. Implementación de uno de los métodos de <i>IERC721</i> correspondiente a la transferencia del token <i>NFT</i>	42
5.4. Creación de una transacción	45
5.5. Fichero para desplegar y relacionar los contratos que heredan entre ellos	46
5.6. Fichero para configurar el despliegue en una <i>Blockchain</i> real	46

Índice de tablas

3.1. Submúltiplos del <i>Ether</i>	16
6.1. Comparativa entre SCH, Wallapop y Opensea	49

Introducción

En primer lugar me gustaría agradecer a los tutores de este proyecto su colaboración, por lo tanto, muchas gracias por su contribución a Jesús Maudes Raedo y Sandra Rodríguez Arribas por parte de la Universidad de Burgos. Además, por parte de HP SCDS a Julia Zuara Jiménez y a Daniel López Palomo por proponer este maravilloso proyecto y su contribución a lo largo de su desarrollo.

Second Hand Chain es un proyecto que gestiona la compraventa de teléfonos móviles de segunda mano, implementando características que aporten valor a los distintos actores en el mercado a lo largo de la vida de un dispositivo.

Para ello se emplea la tecnología *blockchain* mediante el uso de *smart contracts* para gestionar de forma transparente las diferentes transacciones, permitiendo que esta información sea pública e inmutable por ningún tercero debido a las características de diseño de estas. Esta tecnología aporta a los usuarios la garantía de que esos datos no han sido alterados, resolviendo una de las grandes preocupaciones de los usuarios que realizan acciones de compraventa de objetos de gran valor en el mercado de segunda mano.

Además el proyecto implementa una aplicación web que permite interactuar con el *smart contract*, haciendo accesible el proyecto a la mayoría de usuarios. Esta web actualmente está **desplegada**¹ y permite interactuar con el *smart contract* a cualquier usuario que acceda a ella.

Dicha web, permite a los usuarios el registro, la puesta a la venta y la compra de teléfonos móviles además de el cambio de precio o la retirada

¹<https://second-hand-chain.vercel.app/>

de la venta. Debido a las características de *blockchain* permite a los usuarios disponer una gran cantidad de la información de gran valor debido precisamente a la inmutabilidad de esta información.

Actualmente el *smart contract* se encuentra desplegado en una *testnet* de *ethereum*² llamada *Sepolia* que se comporta exactamente igual que la red principal y permite que los usuarios y especialmente desarrolladores puedan probar el *smart contract* sin el pago de *gas fees*³, al poder obtener de forma gratuita la criptomoneda de la red y evitando el costoso despliegue en la red principal, que dependiendo del tamaño de los contratos puede llegar a los miles de euros.

1.1. Estructura de la memoria

- **Objetivos del trabajo:** En esta parte se van a explicar cuales son los objetivos en este proyecto.
- **Conceptos teóricos:** En esta sección se va a explicar cual es el funcionamiento de las redes *blockchain*, para ello se empezara explicando el caso de *Bitcoin* siguiendo su evolución hasta *Ethereum* y los *smart contracts*. Se busca explicar por que estas redes son inmutables y por ello perfectas para el proyecto.
- **Técnicas y herramientas:** En este apartado se va a describir las herramientas utilizadas y el motivo de su elección, exponiendo sus ventajas y desventajas respecto a otras alternativas.
- **Aspectos relevantes del desarrollo:** En este capítulo se explicará las decisiones tomadas durante el diseño y el desarrollo del proyecto además de otros aspectos que son relevantes.
- **Trabajos relacionados:** En esta parte se buscará que proyectos hay similares y una comparación entre las características de cada uno.
- **Conclusiones y líneas futuras:** Finalmente se explicará que características nuevas puede añadir el proyecto. Además repasaremos todas las lecciones aprendidas durante el desarrollo del mismo.

²Red blockchain empleada por desarrolladores y aplicaciones en pruebas que permite no pagar por las transacciones, a diferencia de la red principal

³Costes por realizar transacciones en Blockchain

Objetivos del proyecto

2.1. Objetivos teóricos

1. Estudiar el funcionamiento de redes *blockchain* y su aplicación en entornos de compraventa.
2. Crear un *smart contract* capaz de satisfacer los objetivos técnicos del proyecto.
3. Emplear un sistema que permita mantener la coherencia al resolver el problema de mantener los datos descentralizados gracias a *blockchain* y la imposibilidad por costes de almacenar archivos grandes como imágenes.
4. Crear una aplicación web capaz de aprovechar todas las características del *smart contract*.
5. Investigar y resolver la problemática de desplegar el *smart contract* en una *blockchain* real.
6. Aprender el funcionamiento de la librería de *Javascript React* para desarrollar el *frontend*.
7. Investigar y emplear herramientas para estilar de la forma más óptima posible la web.
8. Emplear la metodología Scrum en la gestión del proyecto.

2.2. Objetivos técnicos

- Seguir el estándar *ERC721* para el desarrollo del *smart contract* que permitiría, si se desease en un futuro, su completa interoperabilidad con otros *smart contracts*.
- El usuario debe ser capaz de registrar teléfonos móviles de forma única por dispositivo, es decir que no se permita que un dispositivo esté registrado varias veces.
- Se debe poder añadir una imagen por dispositivo, asegurando que el almacenamiento de esta siga descentralizado, manteniendo la coherencia con la información guardada en *blockchain*.
- Los usuarios podrán seguir sus transacciones desde *Etherscan*, una web que permite explorar el estado de la *blockchain*.
- La aplicación debe implementar un *marketplace*, capaz de poner teléfonos a la venta, editar su precio, retirarlos de la venta y poder comprarse por otros usuarios.
- Desplegar el *smart contract* en una *testnet* de *Ethereum*.
- Desplegar la aplicación web que interactuará con el *smart contract* desplegado en la *testnet*.
- Para poder realizar transacciones la web debe emplear la *wallet Metamask* que permita de forma segura al usuario firmar con su clave privada las transacciones.
- Realizar un diseño de la aplicación web *responsive* que permita emplear la aplicación desde otros dispositivos como smartphones.
- Emplear el superset de *Javascript*, *TypeScript* desarrollado por *Microsoft* para poder utilizar tipado, mejorando el código y la detección temprana de errores.

2.3. Objetivos personales

- Continuar familiarizándome con el desarrollo web empleando librerías como *React*.
- Investigar y comprender el funcionamiento de la tecnología *blockchain*.

- Aprender a crear una *dApp*⁴.
- Intentar desarrollar la aplicación de la forma lo más descentralizada posible, manteniendo la coherencia de las ventajas de las redes *block-chain*, de forma que la propiedad de los datos no pueda ser alterada por actores a los que no les pertenecen.

2.4. Entregables

- Memoria y Anexos del proyecto.
- Proyecto desplegado en: <https://second-hand-chain.vercel.app/>
- Repositorio público en *Github*: "<https://github.com/axelrg/second-hand-chain>".
- Repositorio privado (Con acceso para el tribunal) en *GitLab*: "<https://gitlab.com/HP-SCDS/Observatorio/2022-2023/secondhandchain/ubu-secondhandchain>".
- Videos para ayudar a comprender el funcionamiento en la web desplegada.
- Tres USB con el proyecto completo.
- Una copia de la memoria en formato físico.

⁴Aplicación descentralizada

Conceptos teóricos

En el caso de este proyecto la gran característica es que almacena sus datos y lógica en *Blockchain*, por ello en los siguientes apartados se explicará el funcionamiento lógico de estas redes para poder comprender cuáles son las grandes características que aportan a este proyecto, así como sus limitaciones. Además se explorará el funcionamiento de *Ethereum*, una *Blockchain* con unas características particulares que permite la ejecución de código.

3.1. Introducción a Blockchain

Blockchain se puede definir como una estructura de datos formada por bloques que contienen transacciones y por sus características es extremadamente complejo y costoso editar transacciones ya realizadas. Esto es debido a que los datos están verificados por sus correspondientes *hashes* y estos a su vez dependen de los hashes de bloques y datos anteriores.

Los pioneros en idear un primer concepto similar a lo que son hoy en día las redes *Blockchain* fueron Stuart Haber y W. Scott Stornetta[20] que buscaban que las marcas de tiempo de los documentos no pudiesen ser modificadas garantizando la autenticidad de estos. Para ello emplearon un sistema que ya empleaba *hashes* para garantizar la consistencia de los datos a lo largo de todo su historial de modificaciones.

Satoshi Nakamoto, es considerado el *padre* de *Blockchain* tras la creación de *Bitcoin* al publicar el primer artículo[25] con la idea de una *Blockchain* moderna. En este trabajo se proponía un sistema descentralizado para gestionar transacciones monetarias sin la intervención de actores como bancos o reguladores que operan en el sistema bancario tradicional.

Este modelo, basado en *proof-of-work* para mantener la coherencia entre los distintos nodos que conforman la red, permite el intercambio de *token-es fungibles* como es la criptomoneda *Bitcoin* pero no de activos más abstractos.

La siguiente gran evolución de *Blockchain* fue llevada a cabo por Vitalik Buterin creador de *Ethereum*. Su propuesta consistía en la capacidad de emplear código en *Blockchain* como describe en el *whitepaper* de *Ethereum*[12], esto permitirá la creación de conceptos tan importantes como smart contract, NFT o dApp.

La tecnología *Blockchain* ha evolucionado de forma muy veloz y pese a que la creación de estas dos redes es relativamente reciente ya existen gran cantidad de proyectos alternativos que ofrecen diferentes características a los usuarios, como pueden ser *Filecoin*, *Monero*, *ZCash*...

3.2. Funcionamiento de Blockchain

En esta sección se va a comentar el funcionamiento de una *Blockchain*. Para ello, se profundizará en el funcionamiento de las estructuras de datos que implementan esta tecnología.

Hash

Un *hash* es el resultado obtenido de una función llamada función *hash* la cual recibe una entrada y la transforma en una cadena resultado de una longitud fija independientemente del contenido de la entrada. Un *hash* será igual a otro sí y solo sí los datos aportados a la función son los mismos en un caso y en otro[44].

Se denomina colisión a que se aporte una entrada distinta a otra y se obtenga el mismo *hash*. Normalmente esto es muy poco frecuente en las funciones *hash* más utilizadas hoy en día, además estas funciones deben ser capaces de calcular el *hash* para una entrada de forma muy veloz.

Las funciones mas empleadas actualmente son las SHA, un estándar propuesto por el NIST⁵ [2], entidad perteneciente al gobierno de Estados Unidos. Por ejemplo, *Bitcoin* emplea SHA-256.

⁵National Institute of Standards and Technology, US Department of Commerce

Merkle Tree

Para estas redes fue clave la invención del *Merkle Tree* [22], un árbol que se emplea para poder obtener un hash único de un conjunto de datos, como se puede ver en la figura 3.1.

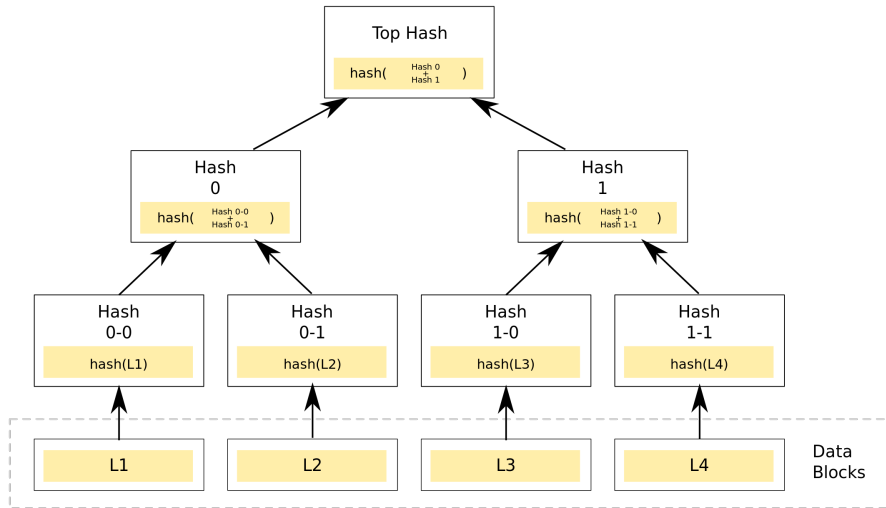


Figura 3.1: Merkle Tree[40]

En este caso tenemos cuatro conjuntos de datos (L1, L2, L3, L4) los cuales componen el conjunto de nodos hoja del árbol, el nodo raíz es el nodo que contiene el *hash* completo del conjunto total. El funcionamiento consiste en que se calculan los *hash* de los datos, representados en los nodos inmediatamente superiores a los hoja.

Estos *hash* se van combinando y se calculan los *hashes* de *hashes* hasta llegar al nodo raíz, el cual contiene un *hash* que nos permite verificar rápidamente la coherencia de todos los *hash* del árbol, ya que al cambiar cualquier dato este *hash* no coincidiría con el previo.

Esta estructura de datos permite la verificación de los datos. En cuanto a complejidad algorítmica podemos considerar que en este caso, similar al de un árbol binario, va a ser para la búsqueda, inserción y borrado:

$$\mathcal{O}(\log n)$$

Mientras que la complejidad espacial será de:

$$\mathcal{O}(n)$$

Criptografía Asimétrica

Las *Blockchain* modernas hacen uso de este sistema para encriptar la información constantemente, por ejemplo, nuestra *wallet* de *Metamask*, necesaria para poder emplear Second Hand Chain está definida por la clave pública y, aunque podamos no ser conscientes, por una clave privada.

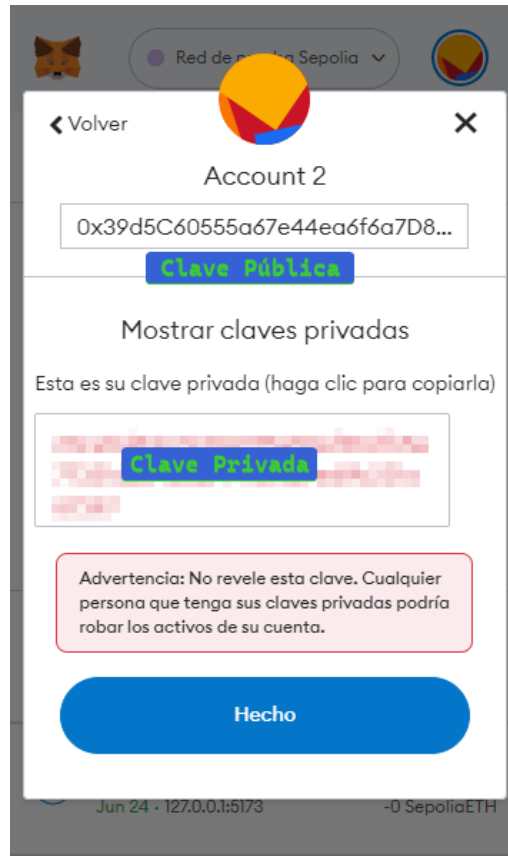


Figura 3.2: Ejemplo de clave pública y clave privada en una de las *wallet* empleadas para crear Second Hand Chain

Este tipo de encriptación hace uso de dos claves distintas [45]:

- **Clave pública:** Esta clave es de dominio público, el propietario la distribuye al público. Permite leer los mensajes encriptados con la clave privada.
- **Clave privada:** Esta clave solamente pertenece al dueño y no debe hacerse pública bajo ningún concepto.

Por lo tanto podemos concluir que:

- Los mensajes encriptados con la clave pública solamente podrán ser descryptados por el propietario de la clave privada.
- Los mensajes encriptados con la clave privada solamente podrán ser descryptados mediante la clave pública y este hecho garantiza que quien lo ha encriptado es el propietario de la clave privada.

Coherencia entre bloques

En este apartado se va a detallar el funcionamiento entre bloques para poder mantener la coherencia y la inmutabilidad. Para ello nos basaremos en una *Blockchain* que sería similar a *Bitcoin* dado que solo sería capaz de registrar transacciones monetarias. A continuación se adjunta una imagen que nos permitirá visualizar mejor este funcionamiento.

The image displays two examples of blockchain blocks, labeled 'Bloque: # 4' and 'Bloque: # 5'. Each block contains a 'Nonce' field, a table of transactions ('Tx'), an 'Anterior' (previous block) hash, and a 'Hash' field. A 'Minar' button is present at the bottom of each block's interface.

Block #4 Data:

Tx	De	De	->	To
\$ 62.19	Rick	->	Ilsa	
\$ 867.96	Captain	->	Strasser	
\$ 276.15	Victor I	->	Ilsa	
\$ 97.13	Rick	->	Sam	
\$ 119.63	Captain	->	Jan Brar	

Anterior: 0000c2c95f54a49b4f2bee7856a7dc3b7c1a408706c848
Hash: 0000c03019ed59586405750968888fb65256e82492480d

Block #5 Data:

Tx	De	De	->	To
\$ 14.12	Denise I	->	Edmund I	
\$ 2,760.25	Lord Gle	->	John Mor	
\$ 413.78	Katherin	->	Miss Auc	

Anterior: 0000c03019ed59586405750968888fb65256e82492480d
Hash: 0000b0aa0cce9eb9432ce5b3081624be9fe47700fe75bf

Figura 3.3: Ejemplo de bloques en una *Blockchain*. Extraído de [11]

Si nos fijamos en fig.3.3 podemos observar dos bloques pertenecientes a una *Blockchain* en los cuales podemos apreciar los siguientes campos:

- **Bloque:** Representa el número de ese bloque, en la figura son 4 y 5.
- **Nonce:** Es un término muy empleado en el ámbito de la criptografía, básicamente será un numero cualquiera que se utilizara para ciertas operaciones criptográficas como veremos en este caso.

- **Tx**: Hace referencia a las transacciones, en este caso fig.3.3 podemos ver que son transacciones económicas entre diferentes personas.
- **Anterior**: Representa el hash del bloque anterior, podemos ver que el anterior del bloque cinco se corresponde con el hash del bloque 4.

El proceso para construir la *Blockchain* será el siguiente:

1. Después de tener una suficiente cantidad de transacciones, este dato varía dependiendo de la *Blockchain* concreta y a veces es incluso dinámico dentro de estas. Se procede a construir el bloque.
2. Para ello se calcula el hash de la combinación de todos los datos en su interior (Anterior, Tx, Nonce e Id).
3. El siguiente paso sería firmar el bloque, esta operación en este ejemplo, consistiría en cambiar el nonce hasta obtener un hash que empiece por cuatro ceros, algo que podemos ver que hay en común entre los distintos hashes, resolver este problema se denomina minar.
4. Una vez firmado el bloque es válido para añadirse al resto de la *Blockchain*, de esta forma se garantiza la autenticidad y la inmutabilidad de tanto los datos de este bloque como los anteriores.

El proceso de minado, por ejemplo en *Bitcoin*, consiste en que el valor del *hash* sea menor a uno llamado *target* que se decide en común entre todos los diferentes nodos que componen la *Blockchain*. Además este valor se va actualizando cada cierto tiempo, de forma que se mantiene la estabilidad de la red al recalcularse la dificultad del problema a resolver cada cierto tiempo. De esta forma se puede garantizar que el tiempo que llevará en firmarse un bloque en *Bitcoin* será siempre similar.

Por ello es muy complicado introducir cambios maliciosos ya que intentar editar un dato previo nos llevaría a tener que minar de nuevo todos los bloques creados desde entonces, lo cual es extremadamente costoso.

Firma de la transacción

Como hemos visto anteriormente los datos en un bloque van a consistir en transacciones, el problema es que tal y como está en la fig.3.3 no tenemos ninguna forma de garantizar que esas transacciones son realizadas por el propietario de ese dinero.

Para resolver este problema se emplea un sistema de clave pública, clave privada. Los emisarios y receptores de la transacción van a ser claves públicas y, además de la transacción, se va a añadir la firma que nos permite, empleando la clave pública verificar que la transacción y la firma sean las mismas después de descryptarlas. De esta forma garantizamos que el emisor del mensaje es el poseedor de la clave pública.

Coherencia y honestidad entre nodos

Un nodo es básicamente un ordenador que dispone de la *Blockchain* y ejecuta un software que garantiza el consenso con el resto de miembros de la red.

Problema de los generales Bizantinos

Este problema[39] consiste en que existe un conjunto de ejércitos que para poder conquistar una ciudad deben actuar de forma sincronizada, ya que existen dos movimientos posibles de los ejércitos en el teatro de operaciones:

- **Ataque**
- **Retirada**

Además existen dos graduaciones militares:

- **Comandante**: Solo hay uno y es el encargado de dar la orden final.
- **Teniente**: Son los encargados de transmitir la orden del comandante y hay varios.

El problema es que hay tenientes traidores y leales, unos transmitirán la orden de forma correcta y otros no.

Este problema aplicado a *Blockchain* nos lleva a relacionar los nodos con los tenientes y a la ciudad con el bloque. Los mecanismos de consenso que hay a continuación son los responsables de resolver este problema al hacer pública la decisión de cada nodo y tomar finalmente la decisión mayoritaria.

Protocolos de consenso

Esta red distribuida va a depender de que los diferentes nodos actúen de forma honesta. Para ello existen diferentes protocolos, siendo los dos más conocidos:

- ***Proof of Stake***^[41]: Se traduciría como prueba de apuesta. Para ser un nodo validador en la red tienes que depositar una determinada cantidad de *tokens*, por ejemplo en *Ethereum* el mínimo son 32ETH, el equivalente a 60.734 USD, si aportas más tienes más posibilidades de generar bloque, lo cual se recompensa. Este sistema se basa en que si posees gran cantidad de tokens no vas a querer mentir y estropear el consenso entre los distintos nodos de la *Blockchain* ya que estarías perdiendo una gran cantidad de dinero.
- ***Proof of Work***^[42]: Se traduciría como prueba de trabajo, el trabajo a realizar es el problema explicado en el caso de *Bitcoin* (calcular un *nonce* válido para firmar el bloque), es muy costoso computacionalmente, y en cierta medida el minero que obtiene el bloque, y por tanto la recompensa, lo hace por suerte. Por ello, el resto de nodos van a aceptar el añadir ese bloque, ya que para aceptar otro tendría que aparecer una solución con un valor menor. No aceptarlo, es indicador de que no eres un nodo honesto, ya que como minero tu objetivo va a ser ir a por otro bloque y conseguir la recompensa.

Por lo tanto podemos concluir que la estabilidad de estos protocolos va a depender de que exista una gran cantidad de nodos, para que los nodos maliciosos nunca sean mayoría en relación a los que son honestos.

3.3. *Ethereum*

Esta *Blockchain*^[15] permite la ejecución de código en *Blockchain* de forma mucho más compleja que las que había hasta el momento de su creación. Para ello emplea la EVM⁶.

La criptomoneda de esta *Blockchain* es el ETH⁷ y pese a que la gran innovación de esta *Blockchain* respecto a las anteriores es la posibilidad de gestionar una gran cantidad de activos, es necesario seguir manteniendo una criptomoneda nativa para poder gestionar el pago de *gas fees* o recompensar a los mineros.

⁶Ethereum Virtual Machine

⁷Ether

Smart Contracts y Dapp

La gran diferencia respecto a las *Blockchains* previas es que permite subir los denominados *smart contracts*, que básicamente son programas que quedan subidos y los diferentes usuarios pueden llamar a sus funciones según deseen.

En *Ethereum* se emplea el lenguaje *Solidity* para programar *Smart Contracts*, estos suelen formar parte de una *Dapp* para la cual actuarían como si se tratase de un *backend* para un aplicación web tradicional o *web2*.

Web3

Otro concepto nuevo al que abre las puertas *Ethereum* es *web3*^[7] que basándose en las características de *Blockchain* pretende aportar las siguiente mejoras a *web2*:

- Los datos no pueden ser alterados, o es extremadamente complicado.
- Que el acceso a los datos sea libre.
- El acceso a los datos no puede ser censurado por nadie.
- Los pagos son en criptomonedas, haciendo innecesaria la intervención de terceros como bancos.

Sin embargo, existen también desventajas:

- Mayor coste, solamente desplegar unos *Smart Contracts* como los que forman Second Hand Chain en la *mainnet* de *Ethereum* puede tener un coste de cientos a miles de euros.
- Mayores dificultades para hacer el *frontend* de la aplicación, en el caso de este proyecto, ya es mas complicado al tener que integrar *Metamask*.
- Los mayoría de usuarios no están acostumbrados al funcionamiento de *web3* y hay conceptos complejos de entender para el usuario estándar sin conocimientos de criptografía o algoritmos, lo cual lleva a que sea más fácil cometer estafas para delincuentes.

Gas Fees

Para poder recompensar a los nodos, por cada transacción los usuarios deben pagar unas *gas fees*[14] que se podrían extrapolar a el pago de impuestos para poder mantener la *Blockchain* operativa.

Además incentiva a los usuarios a solamente subir *smart contracts* que funcionen bien, evitando la subida de código incorrecto o con errores.

La unidad que se emplea para el cálculo de *gas fees* en *Ethereum* es el *gwei* que es una unidad del *Ether*.

Unidad	Valor en Wei
<i>Wei</i>	10^0 <i>Wei</i>
<i>Kwei</i>	10^3 <i>Wei</i>
<i>Mwei</i>	10^6 <i>Wei</i>
<i>Gwei</i>	10^9 <i>Wei</i>
<i>Szabo</i>	10^{12} <i>Wei</i>
<i>Finney</i>	10^{15} <i>Wei</i>
<i>Ether</i>	10^{18} <i>Wei</i>

Tabla 3.1: Submúltiplos del *Ether*

Podemos ver las tres unidades más empleadas en la Tabla 3.1 los cuales son:

- *Wei*: Es la unidad más pequeña y los contratos que guardan precios, como los de Second Hand Chain, los guardan en esta unidad.
- *Gwei*: Es la unidad empleada para calcular los *gas fees* de una transacción.
- *Ether*: Es la unidad estándar para decir los precios de un bien en esta divisa. En Second Hand Chain, pese a guardarlos en *Wei*, todos son mostrados en el *frontend* en *Ether*.

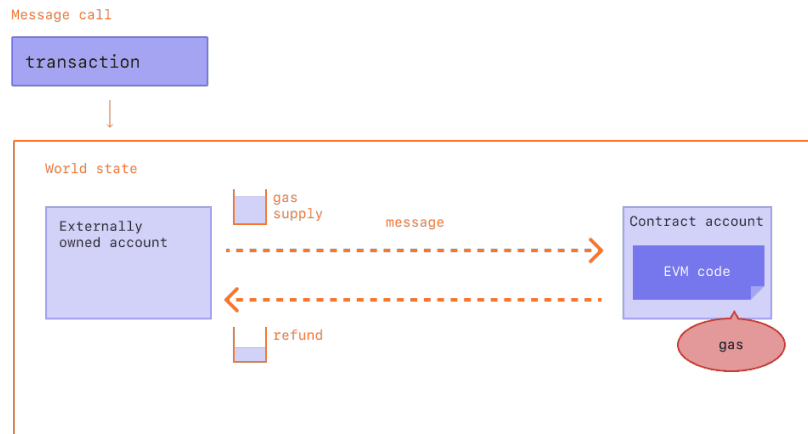


Figura 3.4: Diagrama de funcionamiento de las *gas fees* para una transacción. Extraído de: [14]

Cuando se solicita realizar una transacción, se nos va retirar de la *wallet* el importe que lleve aparejado esa transferencia y además los *gas fees*, los cuales van a calcularse de la siguiente forma:

1. El valor va a depender directamente del bloque en el que se inserte la transacción, cada uno lleva aparejado un valor que cambia dinámicamente dependiendo de la carga de transacciones del bloque anterior. Este valor se denomina *baseFeePerGas* y se va a quemar, es decir, no se lo lleva nadie, ni el minero del bloque.
2. Además vamos a necesitar pagar al minero con lo que se denomina *maxPriorityFeePerGas*, por lo que este valor tendrá que ser de al menos 1 *Wei*, normalmente será muy superior a eso para que a este le compense añadir nuestra transacción al bloque. El *maxPriorityFeePerGas* se lo quedará el minero (el nodo agraciado con el bloque).
3. El coste final total será la suma de estos dos valores (*maxPriorityFeePerGas* y *baseFeePerGas*) multiplicado por el coste de la transacción a realizar, todo ello empleando la unidad *Gwei*.




③ Fee Recipient:	0x1E2cD78882b12d3954a049Fd82FFD691565dC0A5  in 12 secs
③ Block Reward:	0.058890380733411026 ETH (0 + 0.058928488256604711 - 0.000038107523193685)
③ Total Difficulty:	17,000,018,015,853,232
③ Size:	193,723 bytes
③ Gas Used:	23,240,305(77.47%)  +55% Gas Target
③ Gas Limit:	30,000,000
③ Base Fee Per Gas:	0.00000000001639717 ETH (0.001639717 Gwei)
③ Burnt Fees:	 0.000038107523193685 ETH
③ Extra Data:	Illuminate Democratize Distribute (Hex:0x496c6c756d696e61746520446d6f63726174697a6520447374726962757465)

Figura 3.5: Costes de *gas fees* para un bloque real de *Ethereum Sepolia*. Extraído de [17].

En la figura 3.5 podemos ver todos los conceptos anteriores en un entorno real en la *testnet Sepolia* de *Ethereum*. En este caso se está representando los datos obtenidos de un bloque ya minado. Podemos destacar:

- *Fee Recipient*: La cuenta a la que se transfieren todas estas tarifas hasta que el bloque sea minado.
- *Block Reward*: Recompensa al minero del bloque.
- *Size*: El tamaño en bytes del bloque
- *Gas Used*: El máximo de gas dispuesto a pagar por una transacción.
- *Base Fee Per Gas*: Es concepto descrito anteriormente.
- *Burnt Fees*: Valor total del conjunto de *gas fees* que se quema.

3.4. *IPFS y Filecoin*

IPFS[8] es una red distribuida enfocada en el almacenamiento de datos. Se basa en el concepto de tabla *hash* distribuida, de forma que en vez de guardar los datos en una ruta predefinida, esta ruta va a ser el *hash* de esos datos, haciendo que estos datos no puedan ser editados, ya que cambiaría el *hash* y por tanto su dirección (recordemos el concepto de Merkle Tree o Hash Tree). Este *hash* completo se denomina CID y en este trabajo se emplea en

el guardado y lectura de las imágenes (o gifs) de teléfonos a través de un servicio basado en esta tecnología y en Filecoin. El problema de IPFS es que necesita que se guarden los accesos a los *hashes* en un nodo que participe en la red y además los datos no permanecen ya que no se ofrece incentivo a los nodos.

Por esta razón se hace necesario emplear una *Blockchain* como Filecoin[6] está basado en *IPFS*, con la diferencia que implementa una blockchain y recompensa a los nodos por almacenar información, pero de forma que reciben más recompensa al guardar información durante más tiempo, asegurando que van a seguir prestando su servicio.

En Second Hand Chain se emplea un servicio llamado *NFT.Storage*[5] que hace uso de estas dos tecnologías para resolver el problema de los grandes costes de *Ethereum* a la hora de guardar imágenes o otros archivos con mayor tamaño.

Técnicas y herramientas

En este capítulo vamos a ver las metodologías, técnicas y herramientas empleadas en este proyecto como pueden ser los diferentes lenguajes empleados, librerías, patrones de diseño o servicios de terceros.

4.1. Metodologías de gestión de proyectos

Durante el desarrollo de este trabajo se ha empleado la metodología *Scrum*[28]. Esta metodología consiste en ir haciendo mejoras progresivas al producto en lo que se denomina como *sprints*. En cada *sprint* existen tareas a completar, el conjunto de todas ellas conforman el *sprint backlog*. Además en esta metodología se realizan diferentes reuniones:

- *Daily*: Se produce diariamente y los desarrolladores tienen obligada la asistencia.
- *Sprint Review*: Se comentan las diferentes tareas realizadas en el *sprint* por los diferentes miembros.
- *Sprint Retrospective*: Se comparten las diferentes impresiones de los diferentes miembros.
- *Sprint Planning*: Se deciden las tareas a realizar durante el *sprint*.

Además existen los siguientes roles:

- *Product Owner*: Va a establecer prioridades para las diferentes tareas, además tiene la visión completa del producto a desarrollar.

- *Scrum Master*: Su labor consiste en ayudar tanto a desarrolladores como al *Product Owner* a tomar decisiones .
- *Developers*: Son los encargados de llevar a cabo las diferentes tareas.

En nuestro caso, se ha organizado por *sprints* de dos semanas aproximadamente donde se han ido sacando las distintas tareas adelante. Por cada finalización de sprint, se realizaba una reunión con los distintos tutores para comentar las distintas tareas realizadas y las tareas a realizar además de comentar posibles elementos bloqueantes para el desarrollo.

GitLab para gestión *Scrum*

Para la gestión de las diferentes tareas se ha empleado *GitLab*, ya que permite llevar la gestión de proyectos desde el mismo sitio que el control de versiones, lo cual resulta muy cómodo al poder enlazar tareas y *commits*.

No obstante también he encontrado otras características que juegan en su contra como su plan de pago, que es demasiado agresivo y limita a los usuarios gratuitos el poder acceder a elementos tan importantes como por ejemplo los *Burn-down Charts*⁸.

4.2. Patrones de diseño

Durante el desarrollo de este proyecto se han usado algunos patrones de diseño que son descritos a continuación.

Patron *Facade*

Este patrón de diseño [29] consiste en que se proporciona una serie de métodos que permiten de forma sencilla interactuar con un conjunto de gran complejidad que se encuentra detrás de estas.

Un ejemplo en nuestro proyecto es la implementación de la interfaz *ERC721* que interactúa con el resto de métodos del contrato Second Hand Chain y de el mismo (*ERC721*) proporcionado unos métodos estandarizados que permiten de forma sencilla la gestión del *NFT*.

⁸Gráfica que permite evaluar el avance del equipo

Patrón de optimización en el uso de *gas fees*

Este patrón es aplicable exclusivamente a las redes *Blockchain* y consiste en optimizar lo máximo posible el pago de *gas fees* para los usuarios. Como hemos visto en capítulos anteriores, estos costes pueden ser bastante representativos para los usuarios.

```
struct Phone {  
    uint256 id;  
    string model;  
    string brand;  
    string colour;  
    uint16 ram;  
    uint32 mem;  
    address[] owners;  
    uint[] saleTime;  
    uint[] salePrice;  
    uint price;  
    string url;  
}
```

Figura 4.1: *Struct* que representa un teléfono en Second Hand Chain

Como podemos ver en la imagen 4.1 se emplean diferentes tipos para cada dato del *struct*:

- *Id*: Para este dato se emplea el tamaño mas grande posible ya que podría darse el caso en el que al llenarse todos esos ids no se pudiesen crear más teléfonos o se sobre escribiesen los primeros.
- *ram*: Los usuarios van a guardar este dato en Gb por lo que no hacen falta tantos bits.
- *mem*: Como el almacenamiento es un valor en el que se pueden necesitar valores más altos he escogido uint32.

- *precios*: Estos valores los he dejado en `uint256` ya que realmente no se sabe cómo va a funcionar la inflación o la deflación en ETH a largo plazo, de hecho se ha demostrado un activo muy volátil.

Aun así hemos podido ahorrar bastante espacio en memoria adaptando los tipos al dato que realmente van a guardar. En este caso la ganancia es muy importante, ya que se va a multiplicar por todos los teléfonos que se guarden.

Patrón *memory array building*

Este problema[4], figura 4.2, se da, por ejemplo, cuando queremos recuperar los datos de los teléfonos a la venta. La resolución lógica fuera de *Blockchain* sería crear una *linkedList*[1], esto nos permitiría realizar inserciones y borrados en $\mathcal{O}(1)$ manteniendo siempre nuestra lista actualizada al quitar o poner a la venta un teléfono.

```

Phone[] public phones;

mapping(uint => address) phoneOwner;
mapping(address => uint) phonesPerOwner;

function getAllPhonesInAddress(
    address _address
) external view returns (Phone[] memory) {
    Phone[] memory _phones = new Phone[](phonesPerOwner[_address]);
    uint j = 0;
    for (uint256 i = 0; i < phones.length; i++) {
        if (phoneOwner[i] == _address) {
            _phones[j] = phones[i];
            j++;
        }
    }
    return _phones;
}

```

Figura 4.2: *memory array building* en Second Hand Chain

El problema que surge en *Ethereum* es que crear esa lista y mantenerla cuesta dinero. Mientras que iterar en datos ya escritos no, al poder obtenerlos directamente de nuestro nodo local en funciones tipo *view*.

Debemos tener en cuenta que por ejemplo en Second Hand Chain se debe filtrar por teléfonos a la venta y por teléfonos propiedad de una cuenta. Esto si se implementase mediante múltiples listas llevaría a un borrado y escritura de datos muy elevado. Mientras que con este patrón sacrificamos complejidad temporal a cambio de reducir los *gas fees*.

Patrón *Hook*

Este patrón se aplica en los distintos componentes de *React*. Los *hooks*^[3] permiten reutilizar código en toda la aplicación. Su principal función es manejar el estado de los componentes, este estado va a cambiar según se presione un botón, se reciba una respuesta de una consulta a nuestro nodo y en muchos otros casos. Los *hooks* empleados en la aplicación son:

- *useEffect*: Empleado principalmente para las llamadas a nuestro nodo de *Ethereum*, permite ejecutar una porción de código cuando se dan unos cambios.
- *useState*: Maneja el estado de una variable, el estado cambia mediante su función *set* asociada y podemos recuperar el valor en cualquier momento.
- *useRef*: Empleado para hacer referencia a elementos del DOM⁹
- *custom Hooks*: Son Hooks personalizados, no vienen predefinidos como los tres anteriores. En la aplicación se emplean tres para recuperar los móviles.

4.3. Herramientas para el desarrollo del *backend*

Como ya sabemos, la principal particularidad de este trabajo es el hecho de que su *backend* se encuentra en *Blockchain*. Por este motivo, no solo se condicionan las metodologías de programación o de diseño si no que también lo hace con las herramientas empleadas.

Solidity

Para el desarrollo de los contratos inteligentes se ha empleado este lenguaje ^[32] de programación definido por las siguientes características:

- Es un lenguaje de alto nivel.
- Dispone de orientación a objetos.

⁹Document Object Model

- Tiene tipos estáticos, característica requerida debido a la necesidad de garantizar la inmutabilidad en *Blockchain*, además este hecho ayuda a mantener la pulcritud en el uso de espacio en memoria.
- Permite el uso de librerías.

Truffle Suite y Ganache

Este entorno de desarrollo permite migrar contratos a una *Blockchain*, además con su consola dedicada permite interactuar con los contratos inteligentes [34]. Emplea el gestor de paquetes *NPM*.

Ganache forma parte de esta *Suite* y permite simular una *Blockchain* en un entorno de desarrollo local. Dispone de versión *GUI*¹⁰ aunque en este proyecto no se ha usado mucho, ya que me parece más cómodo utilizarlo directamente por consola.

Esta herramienta fue muy usada al principio del desarrollo en conjunción a *Ganache* para poder ejecutar los contratos en local, no obstante fue perdiendo importancia al final del desarrollo del *backend*, este en parte provocado por las grandes ventajas que ofrecen los proveedores de nodos como *Alchemy* en relación a este sistema.

Alchemy

Este servicio satisface una de las necesidades básicas para interactuar con una *Blockchain* real, el disponer de un nodo en la red[10].

Es uno de los denominados proveedores de nodos, como veremos más adelante se tomó la decisión de emplear este, por sus ventajas durante el desarrollo y la facilidad para poner en producción una aplicación descentralizada, algo bastante complejo para un usuario desarrollando por su cuenta si no usara esta tecnología.

Nft.Storage

Para resolver el problema de almacenamiento de archivos de gran tamaño en *Blockchain* se empleó este servicio que permite almacenar los archivos empleando *IPFS* y *Filecoin*, nos permite guardar archivos de forma descentralizada y gratuita.

¹⁰Graphic User Interface

Además, como se ha explicado anteriormente garantiza que estos archivos tampoco se modifiquen.

Emplear un sistema para garantizar las dos condiciones expuestas anteriormente es prácticamente algo obligado para mantener la coherencia y las ventajas de la tecnología *Blockchain*.

4.4. Herramientas para el desarrollo del *frontend*

En este apartado se comentarán las herramientas empleadas en el desarrollo del *frontend*.

TypeScript

Este *superset*^[35] de *JavaScript* permite emplear tipos estáticos. La idea es que el código escrito en *TypeScript* se compila a *JavaScript*, pero al haber empleado las reglas de *TypeScript* para garantizar el uso de tipos estáticos este código en *JavaScript* nos va a garantizar que tiene una total resistencia a en empleo de variables de tipo incorrecto.


```
struct Phone {  
    uint256 id;  
    string model;  
    string brand;  
    string colour;  
    uint16 ram;  
    uint32 mem;  
    address[] owners;  
    uint[] saleTime;  
    uint[] salePrice;  
    uint price;  
    string url;  
}
```

Figura 4.3: Ejemplo de uso de tipos en Second Hand Chain

Podemos ver un ejemplo de este funcionamiento de los tipos de *TypeScript* aplicado en este trabajo en la figura 4.3

Como desventajas podemos considerar, que para ciertos desarrolladores de *JavaScript* puede ser más complicada la adaptación. Aunque en mi caso ha resultado ser lo contrario ya que estoy más acostumbrado a lenguajes con tipos estáticos, como *Java*, *C#* o *C* y otros que también he empleado bastante como *Python* igualmente permiten definir tipos a ciertas variables.

React

Se emplea para crear las interfaces y toda la lógica relacionada con estas. Siguiendo el patrón de diseño MVC, de gran importancia en el desarrollo web, el ámbito de *React* se reduciría solamente a ser la vista. Otras características de *React* [43] son:

- Está basado en componentes

- Dom virtual, además del Dom del navegador *React* tiene otro auxiliar en el que se interpretan los cambios de estado antes de transmitirlo al principal.
- Las *props* o propiedades es la forma de pasar datos entre diferentes componentes.
- El estado de un componente representa como de encuentra en un determinado instante.
- Los *Hooks* permiten reutilizar gran cantidad de código, se deben declarar en los niveles superiores y en ningún caso dentro de porciones de código que se ejecuten de forma condicional o repetida.

Esta librería de *JavaScript*, pese a que a veces es confundido con un *framework* debido a su gran cantidad de características, permite renderizar los elementos en el *DOM* empleando *JSX* (o *TSX* para *TypeScript*). Básicamente lo que hace es transpilar este código a *TypeScript/HTML*.

A screenshot of a code editor window titled "HTML". The editor has a settings gear icon, a refresh icon, and a delete icon in the top right corner. The code is as follows:

```
1 <!-- Hello world -->
2 <div class="awesome" style="border: 1px solid red">
3   <label for="name">Enter your name: </label>
4   <input type="text" id="name" />
5 </div>
6 <p>Enter your HTML here</p>
```

Figura 4.4: Ejemplo código en *HTML*. Extraído de [21]

```
JSX

1  export const Foo = () => (
2    <>
3      {/* Hello world */}
4      <div className="awesome" style={{ border: "1px solid red" }}>
5        <label htmlFor="name">Enter your name: </label>
6        <input type="text" id="name" />
7      </div>
8      <p>Enter your HTML here</p>
9    </>
10 )
11
```

Figura 4.5: Ejemplo código en *JSX*. Extraído de [21]

En la figura 4.4 podemos ver un sencillo código *HTML*, convertido a *JSX* en la figura 4.5, sus principales diferencias y características son:

- Solamente se puede devolver un elemento, por lo que el conjunto de todos los que formen en componente deben ir envueltos en uno único.
- *JSX* devuelve errores al no cerrar correctamente los elementos.
- Las clases se definen con *className* en vez de *class*.
- Se previenen ataques de inyección [27] ya que se eliminan valores insertados antes de proceder al renderizado del componente.

Vite

Vite[37] es una herramienta que permite compilar nuestros proyectos de forma muy rápida y prácticamente en tiempo real, permitiéndonos ver los cambios en nuestro proyecto instantáneamente.

La alternativa para crear una aplicación *React* habría sido emplear *create React app*, no obstante esta tiene algunas desventajas, como que no es tan apropiada para proyectos medianos o pequeños ya que te añade muchas características que probablemente no se van a usar y además es más lento. Aun así *create React app* también es una alternativa que está bastante bien y se podría haber usado en este proyecto[24].

Otra ventaja de usar *Vite* es que está completamente integrada en *Vercel*, permitiéndonos realizar los despliegues de la aplicación de forma muy rápida.

Vercel

Vercel^[36] nos permite desplegar nuestro proyecto de *Vite* en una web lista para ser utilizada.

Para ello normalmente puedes registrarte con tu cuenta de *GitHub* y desplegar el proyecto a partir de los ficheros ahí desplegados. Además de esta forma quedará sincronizado con tu cuenta de *GitHub* permitiendo que todas las mejoras realizadas en la *branch* de producción se suban directamente al despliegue.

Otra de las razones para emplear esta herramienta es que ofrece gratuitamente el mantenimiento de proyectos personales y que no requieran gran cantidad de carga como es el caso de Second Hand Chain.

Tailwind css

Esta herramienta permite estilar las paginas web de forma rápida y eficiente. Su gran diferencia con otras alternativas como *Bootstrap* es que sus clases están definidas entorno a cualidades y no a componentes. Este hecho permite ofrecer una gran cantidad de configuraciones para los distintos elementos y de eficiencia mucho mayor al generar directamente los estilos sobre *CSS*.

Además permite construir diseños *responsive*¹¹ para dispositivos móviles de forma mucho más eficiente siguiendo principios de limpieza de código al establecer un sistema de *breakpoints*, denominado *mobile-first* para condicionar el uso o no de clases en un elemento ^[31].

DaisyUi

DaisyUi es una librería de componentes ya diseñados para poder emplear. Su principal característica es que está completamente integrado con *Tailwind css* permitiendo una interoperabilidad entre estas las clases predefinidas de ambas librerías.

¹¹Diseño web adaptable a móviles o a otros dispositivos con distinto tamaño de pantalla

```

<div className="card card-compact w-96 shadow-xl"
  <figure>
    <img src={imageR} />
  </figure>

  <div className="card-body">
    <div className="tooltip" data-tip="Model">
      <h2 className="card-title justify-center">{p
    </div>
    <div className="tooltip" data-tip="Brand">
      <h2 className="card-title justify-center">{p
    </div>
    <div className="flex w-full">
      <div className="grid h-20 flex-grow card bg-
        <p>
          <div className="tooltip" data-tip="Stora
            <img
              src={hardDriveIcon}
              alt="Memory"
              className="inline w-2/12"
            />

```

Figura 4.6: Ejemplo uso conjunto de *Tailwind css* y *DaisyUi*.

Podemos ver en la figura 4.6 como se integran en las clases de los elementos en un ejemplo sacado de este mismo proyecto, las pertenecientes a *Tailwind css* y a *DaisyUi* configurando el diseño de ese componente.

Web3.js

Este conjunto de librerías [38] nos permite interactuar con nuestro nodo en *Ethereum*. Se ha empleado para hacer llamadas a los métodos de los contratos inteligentes desplegados en la *Blockchain*.

Además en las consultas que incluyan llamadas se deben firmar las transacciones, para ello necesitaremos la clave privada, con lo cual será necesario llamar a la *wallet* instalada en nuestro navegador.

Metamask

Metamask [23] se podría describir como un gestor de nuestra clave pública y privada, va a ser el encargado de interactuar con el usuario cuando se necesiten firmar transacciones. Además incluye una gran cantidad de características para asegurar la seguridad de ambas claves, por ello solicita contraseñas y frases de recuperación.

Además *Metamask* cuenta con una gran popularidad en la comunidad, por ello es la *wallet* que se ha decidido emplear en este proyecto.

Node.js

Finalmente destacar que se ha empleado este entorno de ejecución de *JavaScript* [26], es especialmente destacable ya que las dependencias anteriores se han instalado y gestionado usando el gestor de paquetes nativo de este entorno llamado *NPM*.

4.5. Control de Versiones

Para gestionar el repositorio y sus diferentes versiones se ha empleado *git* [19]. Este gestor de versiones permite seguir los cambios que ha habido a lo largo del tiempo en el proyecto. Para utilizar *git* lo normal es emplear un servicio de *hosting* para el repositorio, En este caso se han empleado dos:

- *GitLab*: Empleado como el principal durante la mayoría del desarrollo del proyecto, era de necesaria utilización debido a que este proyecto ha sido ofertado por el programa *SCDS* de la empresa *HP*, y su repositorio estaba *hosteado* en este servicio. Se puede acceder (si tienes permisos) en: <https://gitlab.com/HP-SCDS/Observatorio/2022-2023/secondhandchain/ubu-secondhandchain>
- *GitHub*: Principalmente usado al final del proyecto, se migró a este servicio ya que en *GitLab* estaba restringida la posibilidad de hacer público el repositorio al ser propiedad de *HP*. Se encuentra en (Puede acceder cualquiera al ser público): <https://github.com/axelrg/second-hand-chain>

Utilizar un sistema de control de versiones es una característica clave en cualquier proyecto de *software* ya que al tener los ficheros una gran complejidad y existir una gran cantidad de ellos cualquier cambio indeseado puede hacernos perder características y provocarnos el no saber volver a la versión anterior o ver cuales han sido los cambios realizados.

Además tiene una gran influencia en otros aspectos como la metodología *Scrum* ya que facilita el trabajo en equipo y el trabajo en diferentes *features* por distintos desarrolladores al mismo tiempo mediante las *branch* y realizando un *merge* posteriormente.

4.6. Otras herramientas utilizadas

- *Overleaf*: Es un editor de \LaTeX online que se ha empleado para redactar esta memoria y los anexos adjuntos.
- *VScode*: Es un editor de textos que ofrece a los usuarios la posibilidad de añadir extensiones, algunas de ellas amplían tanto la funcionalidad de este software que lo llegan a convertir en un pequeño *IDE*.
- *EtherScan*: Esta web[18] permite el seguimiento de las transacciones registradas en la *Blockchain Sepolia*, prácticamente es un estándar y todas las *dApp* ofrecen seguir el estado de su transacción. Second Hand Chain no es una excepción en esto y podemos seguir las transacciones desde esta web.
- *Brave*: Permite abrir enlaces de *IPFS* directamente, sin necesidad de *gateway*.

Aspectos relevantes del desarrollo del proyecto

Durante este capítulo se van a exponer los aspectos más relevantes que se han producido a lo largo del proyecto. Para ello haré un repaso evaluando lo ocurrido en cada fase.

5.1. Fases iniciales

Nada más ver este *TFG* propuesto por el Observatorio Tecnológico HP SCDS, me encantó la idea original que proponía un *marketplace* de objetos de segunda mano basado en *Ethereum*. En esta primera versión se proponía identificar estos objetos mediante un *QR* que quedaría registrado en *Blockchain*.

Tras investigar bastante sobre el funcionamiento de estas tecnologías, de las que yo no tenía muchos conocimientos inicialmente y a nivel técnico ninguno, me pareció mejor adaptar la idea al mercado de teléfonos de segunda mano, principalmente por estos motivos:

- Registrar los teléfonos da a futuros compradores datos valiosos y verificados como las veces que ha sido vendido el teléfono, el histórico de precios o las características.
- Especializarse en un mercado aporta una gran cantidad de ventajas, ya que puedes ofrecer mejor servicio a los clientes.
- Los teléfonos se pueden identificar por su *IMEI* este es bastante más complicado de cambiar que un *QR* que se puede perder o deteriorar

con el uso. En cualquier caso, para cambiar el *IMEI* tendrías que *rootear* el teléfono, proceso que se puede saber si se ha realizado o no, además es una práctica en claro retroceso.

- Para guardar los objetos en *Blockchain* (aunque realmente en cualquier BBDD) se deben conocer las características del objeto, que se pueda vender cualquier cosa complica enormemente esta tarea y probablemente habría llevado a dejar características demasiado genéricas.
- El mercado de teléfonos de segunda mano se encuentra en claro ascenso, solo hay que ver los precios de los teléfonos actualmente.
- Los clientes gastan grandes cantidades de dinero en estos teléfonos, por lo que invertir en que sus datos sean más fiables para futuros compradores puede ser un coste asumible.

. Inicialmente se definieron una serie de ideas, de forma conjunta con los tutores, sobre las que se fueron construyendo los requisitos de la aplicación:

- La aplicación consiste en un *marketplace* para intercambiar *NFTs* de teléfonos.
- Los usuarios deben poder registrar estos teléfonos.
- Los usuarios podrán vender y comprar teléfonos.
- Habrá un tipo de usuario, sin *wallet* que podrá consultar los teléfonos a la venta.

DIAGRAMA DE SECUENCIAS

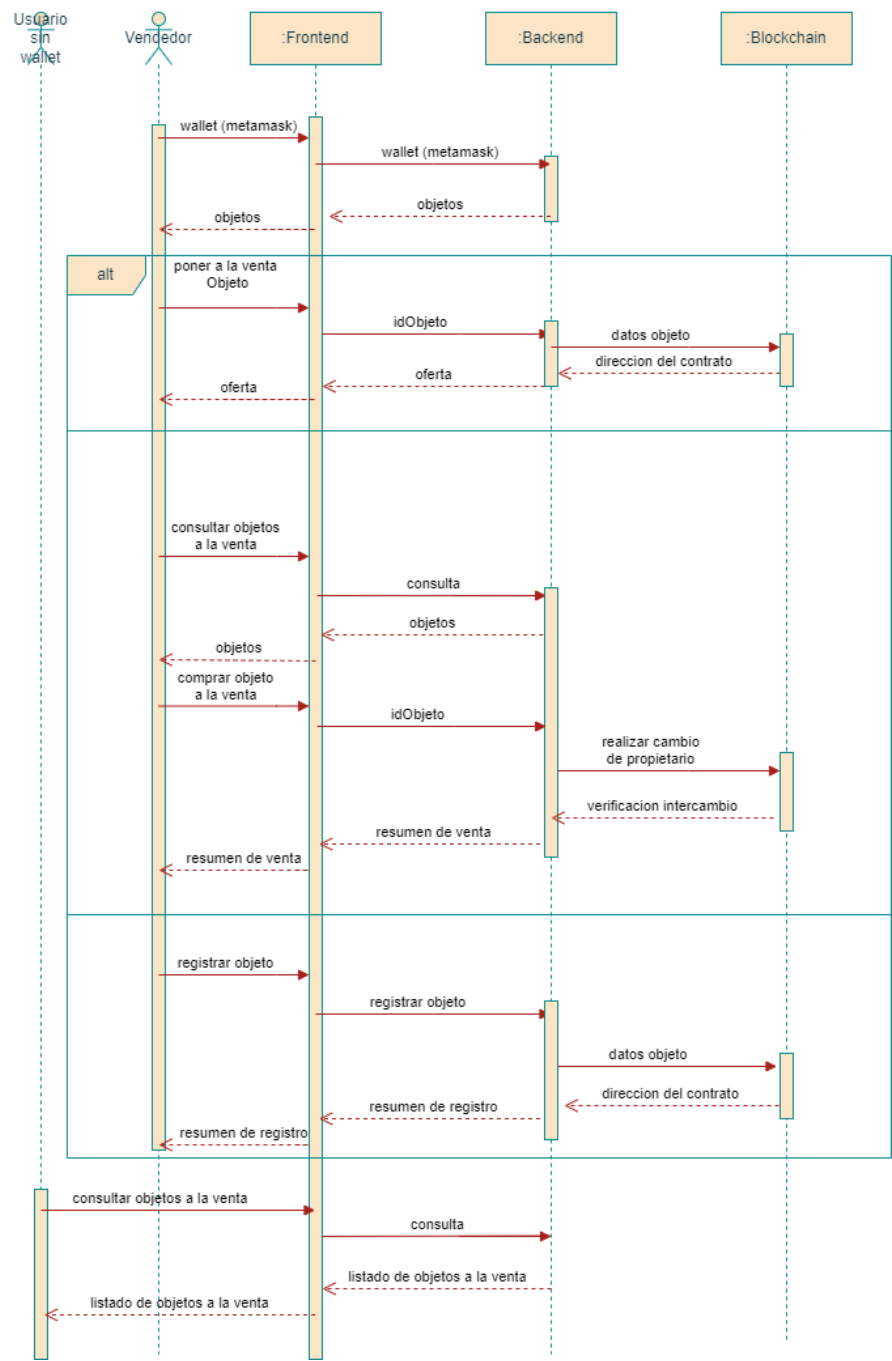


Figura 5.1: Diagrama de secuencias inicial.

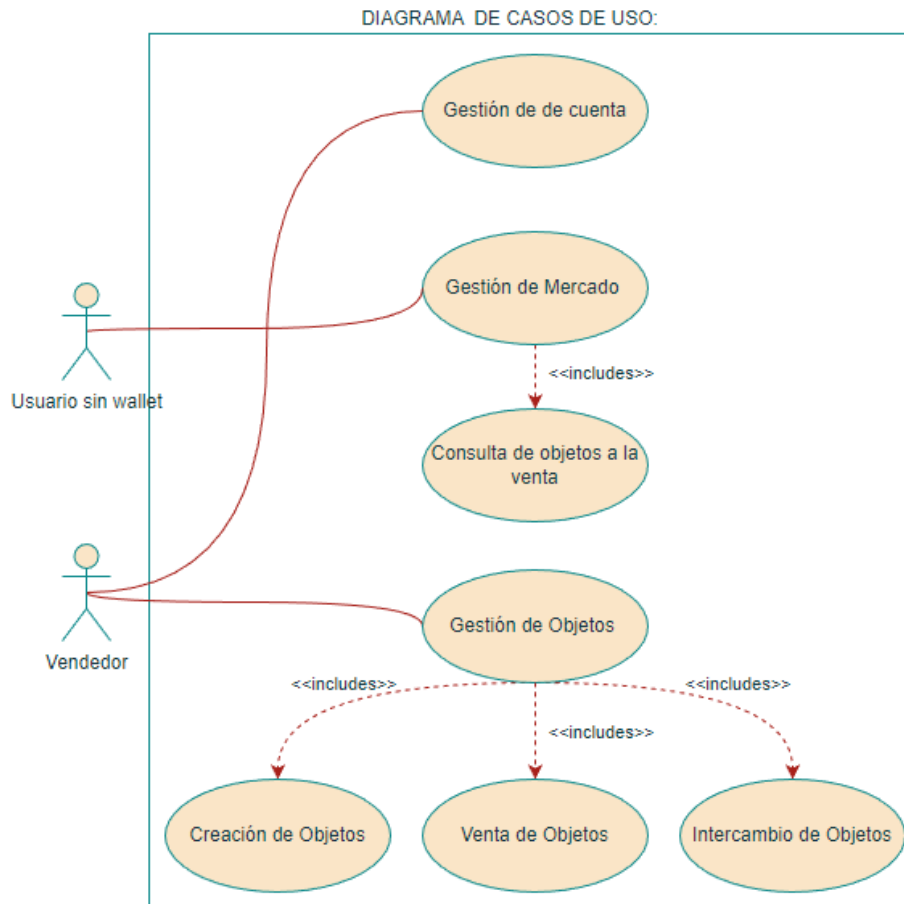


Figura 5.2: Diagrama de casos de uso inicial.

A partir de estas ideas se crearon unos diagramas UML de casos de uso, figura 5.2, y de secuencias, figura 5.1, que cumplían con estos requisitos iniciales. Estos diagramas iniciales se pueden consultar en:

- Diagrama inicial de secuencias: <https://github.com/axelrg/second-hand-chain/blob/main/ModelsUML/sequenceDiagram.drawio>
- Diagrama inicial de casos de uso: <https://github.com/axelrg/second-hand-chain/blob/main/ModelsUML/useCaseDiagram.drawio>

Aunque posteriormente, como en cualquier proyecto real y precisamente por ello se emplea *Scrum*, se fueron añadiendo otros como la posibilidad de retirar de la venta , cambiar el precio, emplear *Etherscan*...

Finalmente comencé con la configuración del repositorio, proceso que me dio algunos problemas al configurar la clave *SSH* empleada para poder hacer *clone* del repositorio de *GitLab* y poder hacer *push*. Durante este proceso configuré las cuentas de forma equivocada, motivo por el cual se pueden ver al principio del proyecto algunos *commits* con cuentas mal escritas o la mía personal.

5.2. Desarrollo *backend*

En este proyecto se definió que desde un primer momento que la aplicación debía estar basada en la *Blockchain Ethereum*. Por ello se hizo necesario aprender *Solidity*, un lenguaje que debido a las particularidades que tiene *Ethereum* incluye unos conceptos que son bastante distintos al resto de lenguajes:

- Modificadores de acceso: Existen modificadores [30] para los métodos que son bastante diferentes a los normalmente empleados:
 - *View*: con este modificador describimos que una función solamente lee datos, no va a tener costes de gas.
 - *Pure*: con este modificador describimos que un método no va ni a leer datos de *Blockchain* ni a escribirlos, puede ser empleada para crear un función de sumar dos números pasados por parámetro.
 - *Public*: El el modificador por defecto, permite acceder a una función internamente o externamente.
 - *Private*: Solo es visible para el contrato y no para los contratos derivados.
 - *External*: Pueden llamarse desde otro contrato y vía transacciones, para llamarlas dentro del contrato habría que usar "this".
 - *Internal*: Solo es visible para el contrato y para los contratos derivados.
 - *Payable*: para funciones que se van a llamar en transacciones a las que se les ha añadido *Ether*.
 - *Memory*: Datos guardados solo localmente.
 - *Storage*: Datos guardados en *Blockchain*.
- Estos contratos van a lanzar eventos cuando se produzcan transacciones.

- Funciones *require*, verifican que se cumplen las condiciones.
- Las listas tienen algunas diferencias, especialmente las de tamaño dinámico.

Por todas estas razones la adaptación a *Solidity* puede resultar algo costosa. Para resolver este problema consulté varios recursos:

- *Cryptozombies*[13]: Es un recurso muy popular para aprender a programar en este lenguaje, la didáctica que emplea es excepcional, quizás la única pega podría ser que está algo obsoleto.
- Guías de *Truffle*[33]: Estos recursos permiten adaptarse a la programación en este lenguaje y al uso de *Truffle*.
- Documentación de *Solidity*.

Otra de las características importantes a destacar es la implementación de los contrato siguiendo el estándar *ERC721*[46], el estándar para *NFTs*. Su implementación es clave ya que este estándar permite que los contratos puedan interactuar entre sí, además terceros podrán adaptar sus códigos fácilmente al saber que lo empleas.

```
function safeTransferFrom(
    address _from,
    address _to,
    uint256 _tokenId) external payable{
    _validationPreTransfer(_from, _to, _tokenId);

    require(msg.value >= phones[_tokenId].price, "No payment no token");

    if (msg.value > phones[_tokenId].price){
        payable(msg.sender).transfer(msg.value - phones[_tokenId].price);
    }

    payable(phoneOwner[_tokenId]).transfer(phones[_tokenId].price);

    _transfer(_from, _to, _tokenId);
    emit Transfer(_from, _to, _tokenId);
}
```

Figura 5.3: Implementación de uno de los métodos de *IERC721* correspondiente a la transferencia del token *NFT*.

A nivel técnico, como podemos ver en la figura 5.3 consiste en la implementación de unos métodos comunes a lo largo de estas aplicaciones que deben implementar la interfaz *IERC721*.

Además incluye unos métodos para dar permisos a un tercero para transferir ese *token*. En el caso de la implementación de Second Hand Chain,

cuando se pone un *NFT* a la venta, para que sea automáticamente transferido en el momento en el que produce el pago al propietario, se ha dado permisos a la dirección del contrato para poder transferirlo. De esta forma el código del contrato garantiza que la transferencia del *token* solo se produce si hay pago.

Esto da una solución al problema de desconfianza generado entre las dos partes en cuanto a que quién haga la transferencia primero puede quedarse sin su objeto de valor. De esta forma, queda garantizado por el contrato digital que esto nunca se va a producir.

Otro de los problemas que fueron resueltos fue el encontrar una red *Blockchain* adecuada para desplegar el ya casi finalizado contrato. Tras buscar bastantes opciones vi que desplegarlo en la red principal iba a ser imposible por los costes de gas. No obstante, *Ethereum* ofrece para los desarrolladores redes de prueba, siendo *Sepolia* la disponible en este momento.

Estas redes a efectos prácticos se comportan exactamente igual que la principal pero sus *Ether* se pueden obtener de forma gratuita. Además, para que no se conviertan en redes *Blockchain* para especular, tienen una vida útil de unos seis años, por lo que *Sepolia* seguirá viva hasta el uno de enero de 2027 al ser creada al final de 2021.

Finalmente, según se iba finalizando el desarrollo de la parte de *backend* iba surgiendo el problema de comunicar el contrato, el cual iba a ser desplegado en la *testnet Sepolia*, con el futuro *frontend*. A raíz de investigar, encontré que para ello iba a ser necesario disponer de un nodo en *Ethereum Sepolia*. Por ello, comparé los diferentes proveedores de nodos[16] y llegué a la conclusión de que *Alchemy* ofrecía el mejor servicio.

El modelo que seguí fue la obtención de una *API Key*[9] de *Alchemy* para tener un dirección a la que mandar las peticiones, que pueden ser solamente de información al nodo, o transacciones firmadas para registrar en *Sepolia*.

En resumen, el objetivo era poder desplegar el contrato en *Blockchain* para poder realizar un *frontend* basado en una red real y que fuese totalmente funcional de cara al usuario que va a probar la aplicación.

No obstante, lograr esto en la red principal, aún habiendo esperado a desplegar la versión final del contrato, habría tenido unos costes prohibitivos. Estamos hablando de miles de euros, entre despliegue de los contratos y los costes de probar la aplicación. Además, los usuarios tendrían que comprar *Ether* para registrar teléfonos y operar con ellos.

Finalmente, cuando se decidió añadir imágenes a las tarjetas surgió el problema de cómo almacenar esas imágenes. Dado que eran ficheros muy grandes, además no hay forma fácil de guardarla en *Sepolia*, ya que los contratos tienen muy limitados los tipos.

Esto nos habría obligado a transformar la imagen a texto, probablemente bit a bit, y de haberlo hecho así nos cerraríamos la puerta a poder escalar la aplicación en un futuro a la *mainnet*, al tener en esta un coste desmesurado el almacenar tantos datos.

La solución encontrada fue emplear el servicio *NFT.storage* que a través de *IPFS* y *Filecoin* permite guardarlas garantizando los siguientes principios:

- Los archivos van a ser inmutables.
- Estos archivos se guardan en una red descentralizada.

Por ello el proceso seguido será:

1. Se suben los archivos a *NFT.storage* que a su vez los almacena en *IPFS* y *Filecoin*.
2. Este servicio nos devuelve el denominado *CID*, que básicamente va a ser un *hash* formado a partir del fichero subido y que va a definir su ruta.
3. Con este *CID* procedemos a crear nuestro *NFT*, de esta forma estamos garantizando que toda la información sea coherente e inmutable en el proceso.
4. A la hora de recuperar la información empleamos el *CID* que forma parte de los datos del *NFT*, podemos acceder a esta información mediante un navegador que admita *IPFS*, cómo *Brave*.
5. Para recuperar la información desde otros navegadores se puede emplear un *gateway* como el que se emplea, propiedad también de *NFT.storage*.

Encontrar una solución a este problema era de vital importancia, ya que realizar parte del *backend* en *Blockchain* y otra parte en una base de datos estándar no me parecía que tuviese mucho sentido, al acabar condicionado por las desventajas que ofrecen unas y otras.

5.3. Desarrollo *Frontend*

Tras terminar con las versiones iniciales de los contratos comencé con el desarrollo del *frontend*, para ello decidí usar *React*, una librería de *Javascript*.

Uno de las mayores complicaciones en el desarrollo del *frontend* fue su integración con los contratos. Para ello decidí emplear una librería llamada *Web3.js*, que permite construir las transacciones, solicitar que sean firmadas a la *wallet* y enviarlas posteriormente al nodo de *Sepolia*.

```
const signPutOnSale = async () => {  
  const tx = {  
    from: selectedAccount,  
    to: contractAddressSHC,  
  
    'data': contract.methods.putPhoneOnSale(  
      web3.utils.toWei(price, 'ether'),  
      id  
    ).encodeABI()  
  }  
  
  try {  
    var response : string = await window.ethereum.request({  
      method: 'eth_sendTransaction',  
      params: [tx]  
    })  
    return response  
  } catch (error) {  
  }  
}
```

Figura 5.4: Creación de una transacción

En la figura 5.4 podemos ver como se define una transacción para poner a la venta un teléfono, en los campos de emisor y receptor se debe definir el emisario del mensaje y el receptor de la transacción. Al ser un método de un contrato, el receptor va a ser la dirección del contrato. Las dos direcciones son claves públicas tanto la del usuario como la del contrato.

Posteriormente llamamos al método que permite que *Metamask* solicite firmar al usuario la transacción y se envíe. Además, al realizar este último proceso esperamos a tener un *hash* de la transacción que permitirá al usuario realizar el seguimiento de esta desde *Etherscan*.

Como todas estas operaciones son asíncronas, hay que trabajar con los condicionantes que ello conlleva, además me resultó bastante complejo encontrar con el formato correcto para construir la transacción al no haber gran cantidad de información respecto a este tema, y aún menos si el despliegue es en *Blockchins* reales.

5.4. Despliegue

Otro de los aspectos relevantes a destacar es el proceso de despliegue de los contratos en *Blockchains* reales, difiere bastante de realizarlo en *Ganache* (en local), donde cuentas con las facilidades que ofrece la consola de *Truffle*.

```
truffle > migrations > 01-secondHandChainDeployment.js > ...
You, 1 second ago | 1 author (You)
1  var SecondHandChain = artifacts.require("SecondHandChain");
2  var ERC721 = artifacts.require("ERC721");
3  var ERC721Metadata = artifacts.require("ERC721Metadata");
4
5  module.exports = async function(deployer, network, accounts) {
6    await deployer.deploy(SecondHandChain);
7    await deployer.link(SecondHandChain, ERC721);
8    await deployer.deploy(ERC721);
9    await deployer.link(ERC721, ERC721Metadata);
10   await deployer.deploy(ERC721Metadata);
11 }
```

Figura 5.5: Fichero para desplegar y relacionar los contratos que heredan entre ellos

Podemos ver en la figura 5.5 en que orden se va a producir el despliegue de los contratos, este debe ser obligatoriamente así, ya que si no, no se van a heredar las variables de uno a otro, lo cual causaría el no funcionamiento de ellos. Considero que este aspecto es reseñable ya que fue uno de los problemas que me encontré, desplegaba los contratos, pero resultaba que después no se comunicaban entre ellos. Esto se debió a que la mayoría de despliegues descritos en diferente documentación no cuentan más que con uno o dos contratos.

```
sepolia: {
  provider: () =>
    new HDWalletProvider(
      'Clave Privada',
      'https://eth-sepolia.g.alchemy.com/v2/Alchemy API Key',
    ),
  network_id: 11155111,
  timeoutBlocks: 200,
},
```

Figura 5.6: Fichero para configurar el despliegue en una *Blockchain* real

En la figura 5.6 podemos ver como debe ser la configuración para el despliegue de la aplicación en *Sepolia*:

- Necesitamos pasar la clave privada de nuestra cuenta, desplegar los contratos tiene un gran coste, por ello hay que asegurarse de disponer de suficiente *Ether*.
- Para mandar esta información al nodo hay que llamarle, para lo cual necesitamos la *API Key*.
- Se especifica el identificador de red de *Sepolia*.
- Número de bloques a ser minados antes de que la transacción caduque.

Este fichero¹², se puede ver que existe en el repositorio hasta más o menos el momento en el que se produce el primer despegue en *Sepolia*, momento en el que, por razones obvias, fue eliminado y añadido a *.gitignore*.

¹²*truffle-config.js*

Trabajos relacionados

Actualmente, el estado del arte de aplicaciones similares a Second Hand Chain es complejo de investigar ya que no he podido encontrar aplicaciones orientadas a la venta de móviles de segunda mano respaldada por la tecnología *Blockchain*. Sin embargo, para la realización de este proyecto se han tomado como referentes dos *marketplaces* referentes hoy en día en el intercambio de bienes entre clientes.

Característica	SCH	Wallapop	Opensea
Basada en Blockchain	SI	NO	SI
Historial de Propietarios	SI	NO	SI
Filtros	NO	SI	SI
Relaciona NFT y objeto físico	SI	NO	NO
Subastas de productos	NO	SI	SI
Filtro por distancia	NO	SI	NO APLICA

Tabla 6.1: Comparativa entre SCH, Wallapop y Opensea

- *Wallapop*: Es un *marketplace* propiedad de una empresa Española con sede en Barcelona, en los últimos años y especialmente a raíz de la pandemia ha experimentado un gran incremento de ventas.
- *OpenSea*: Es un *marketplace* centrado en la venta de arte *NFT*, actualmente es el más popular para comprar este tipo de objetos.

Como podemos ver en la tabla 6.1 estas dos ideas, difieren bastante de lo que intenta aportar Second Hand Chain, no obstante también existen

similitudes, siendo este proyecto un intento de mezclar las ventajas de ambos mundos.

Conclusiones y Líneas de trabajo futuras

Durante este proyecto he podido emplear una gran cantidad de herramientas y tecnologías, por ello, en este último capítulo vamos a repasar todas estas conclusiones y además a estudiar diferentes opciones para mejorar el producto en un futuro.

7.1. Conclusiones

Se ha conseguido cumplir con los objetivos iniciales establecidos al inicio de este proyecto y además se han añadido diversas mejoras a lo inicialmente establecido:

- Se ha conseguido desplegar el proyecto tanto a nivel de *frontend* cómo de *backend*, permitiendo a los usuarios poder interactuar con la aplicación de la forma lo más sencilla posible.
- Creación de unos contratos digitales que satisfacen las necesidades establecidas inicialmente, todo ello respetando unos principios de diseño que garantizan la inmutabilidad de los datos y el libre acceso a ellos.
- He creado un *frontend* que permite satisfacer todas las necesidades propuestas y que es capaz de interactuar con *Blockchain*.
- Se ha implementado un sistema que permite a los usuarios añadir imágenes a los teléfonos, todo ello en coherencia con las grandes

ventajas de *Blockchain* y resolviendo el gran problema del coste de almacenaje de gran cantidad de datos en estos sistemas.

- Mediante el uso de la metodología de gestión de proyectos *Scrum* hemos podido gestionar el proyecto de forma que este se haya podido adaptar a la introducción de mejoras y cambios a los requisitos establecidos inicialmente.
- Todo el proyecto se ha realizado empleando herramientas que pueden ser consideradas *state of the art*, de esta forma se facilita la introducción de mejoras en un futuro y su mantenibilidad .
- He adquirido una gran cantidad de conocimientos tanto a nivel teórico como técnico de *Blockchain*, los cuales desconocía por completo antes del inicio de este proyecto.
- He ampliado mis conocimientos sobre programación web, principalmente en lo que respecta a *React* y el resto de librerías para facilitar el estilado de la web.
- A nivel personal, aunque no contaba como uno de los requisitos iniciales de la aplicación, he conseguido desplegar la aplicación y hacerla accesible a cualquiera, lo cual permite poder enseñar fácilmente tu trabajo.
- Finalmente he descubierto que la tecnología *Blockchain* tiene un gran potencial detrás y un funcionamiento muy ingenioso el cual he disfrutado aprendiendo, al poder aplicar conocimientos estudiados a lo largo del grado.
- No obstante, también he conocido de primera mano sus limitaciones, por las cuales es mucho más complicado que en la *web2* construir aplicaciones y por eso en el estado actual de esta tecnología considero que debe tener un papel auxiliar.
- Por otro lado considero que las ventajas de esta tecnología son complicadas de explicar a la población general sin conocimientos de este tema. Principalmente el por que deben pagar por usar esta tecnología mientras que en una mayoría de servicios de *web2* no.

7.2. Líneas de trabajo futuras

Dado que al final este proyecto tiene una fecha de entrega que hay que cumplir, como al final cualquier otro, se han quedado bastantes ideas sin poder ser implementadas, las cuales expongo a continuación:

- Implementar filtrado de teléfonos, actualmente no se pueden filtrar teléfonos, esta característica sería sin duda la primera que implementaría, debido a su impacto en la UX¹³.
- Los usuarios deberían poder relacionarse con otros usuarios para poder gestionar la transferencia física de los teléfonos, para ello, habría que implementar un sistema que permitiese relacionarse con usuarios cercanos entre si.
- Permitir que los usuarios puedan poner nota a otros usuarios, de forma que permita que la comunidad se modere a si misma. Además, las ventajas de *Blockchain* permiten limitar problemas como el *review bombing*¹⁴, al costar dinero el proceso de poner reseñas o las reseñas falsas al solamente dar permisos al usuario para escribir una reseña del producto que has comprado.
- Añadir en los contratos digitales el almacenamiento de objetos transacción. De esta forma, se puede asignar a estos objetos datos como comentarios sobre el estado de un producto comprado, fotos. Permitiendo crear un histórico del producto con más datos.
- Poder ver datos sobre los propietarios, reseñas recibidas, móviles a la venta, volumen de intercambios. Parte de esta información se podría extraer empleando la integración con *Etherscan*, el objetivo sería ampliar esta información y facilitar su acceso al usuario.
- Desplegar el proyecto a la red principal, cualquier *Dapp* aspira a ser migrada a la red principal que es la que garantiza que los datos sean permanentes, aunque por los motivos expuestos en anteriormente en esta memoria esto tiene unos costes económicos muy grandes.

¹³Experiencia de usuario

¹⁴Fenómeno de escribir reseñas negativas para dañar un producto

Bibliografía

- [1] Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell — bigocheatsheet.com. <https://www.bigocheatsheet.com/>. [Accessed 28-Jun-2023].
- [2] Hash functions standards nist. [Online; accessed 27-June-2023].
- [3] Hooks Pattern — patterns.dev. <https://www.patterns.dev/posts/hooks-pattern>. [Accessed 28-Jun-2023].
- [4] Memory Array Building — fravoll.github.io. https://fravoll.github.io/solidity-patterns/memory_array_building.html. [Accessed 28-Jun-2023].
- [5] NFT.Storage - Free decentralized storage and bandwidth for NFTs on IPFS Filecoin. — nft.storage. <https://nft.storage/>. [Accessed 28-Jun-2023].
- [6] Overview — docs.filecoin.io. <https://docs.filecoin.io/basics/what-is-filecoin/overview/>. [Accessed 28-Jun-2023].
- [7] Web2 vs Web3 | ethereum.org — ethereum.org. <https://ethereum.org/en/developers/docs/web2-vs-web3/>. [Accessed 27-Jun-2023].
- [8] What is IPFS? | IPFS Docs — docs.ipfs.tech. <https://docs.ipfs.tech/concepts/what-is-ipfs/#defining-ipfs>. [Accessed 28-Jun-2023].
- [9] Alchemy. Alchemy Quickstart Guide — docs.alchemy.com. <https://docs.alchemy.com/docs/alchemy-quickstart-guide>. [Accessed 29-Jun-2023].

- [10] Alchemy. What is an RPC node? — alchemy.com. <https://www.alchemy.com/overviews/rpc-node#how-to-run-your-own-rpc-node>. [Accessed 29-Jun-2023].
- [11] Anders94. GitHub - anders94/blockchain-demo: A web-based demonstration of blockchain concepts. — github.com. <https://github.com/anders94/blockchain-demo/>. [Accessed 26-Jun-2023].
- [12] Vitalik Buterin. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.
- [13] CryptoZombies. CryptoZombies - Aprende a programar en Ethereum. Impulsado por Loom Network — cryptozombies.io. <https://cryptozombies.io/es/>. [Accessed 29-Jun-2023].
- [14] Ethereum.org. Gas and fees | ethereum.org — ethereum.org. <https://ethereum.org/en/developers/docs/gas/>. [Accessed 28-Jun-2023].
- [15] Ethereum.org. Introducción a Ethereum | ethereum.org — ethereum.org. <https://ethereum.org/es/developers/docs/intro-to-ethereum/>. [Accessed 27-Jun-2023].
- [16] ethereum.org. Nodes as a service | ethereum.org — ethereum.org. <https://ethereum.org/en/developers/docs/nodes-and-clients/nodes-as-a-service/>. [Accessed 29-Jun-2023].
- [17] etherscan.io. Sepolia Blocks 3785657 | Etherscan — sepolia.etherscan.io. <https://sepolia.etherscan.io/block/3785657>. [Accessed 28-Jun-2023].
- [18] etherscan.io. TESTNET Sepolia (ETH) Blockchain Explorer — sepolia.etherscan.io. <https://sepolia.etherscan.io/>. [Accessed 29-Jun-2023].
- [19] git scm. Git — git-scm.com. <https://git-scm.com/video/what-is-version-control>. [Accessed 29-Jun-2023].
- [20] Stuart Haber and W Scott Stornetta. *How to time-stamp a digital document*. Springer, 1991.
- [21] Ritesh Kumar. HTML to JSX — transform.tools. <https://transform.tools/html-to-jsx>. [Accessed 29-Jun-2023].
- [22] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

- [23] Metamask. Home | MetaMask developer documentation — docs.metamask.io. <https://docs.metamask.io/>. [Accessed 29-Jun-2023].
- [24] musabdev. Create React App vs Vite: Choosing the Right Build Tool for Your Project — dev.to. <https://dev.to/musabdev/create-react-app-vs-vite-choosing-the-right-build-tool-for-your-project-3ni1>. [Accessed 29-Jun-2023].
- [25] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.
- [26] nodejs.org. Node.js — nodejs.org. <https://nodejs.org/es>. [Accessed 29-Jun-2023].
- [27] reactjs.org. Presentando JSX – React — es.legacy.reactjs.org. <https://es.legacy.reactjs.org/docs/introducing-jsx.html>. [Accessed 29-Jun-2023].
- [28] Sakshi Sachdeva. Scrum methodology. *Int. J. Eng. Comput. Sci*, 5(16792):16792–16800, 2016.
- [29] Douglas C Schmidt. Wrapper facade. *C++ Report*, February, 1999.
- [30] solidity. Contratos x2014; documentacixF3;n de Solidity - UNKNOWN — solidity-es.readthedocs.io. <https://solidity-es.readthedocs.io/es/latest/contracts.html#visibilidad-y-getters>. [Accessed 29-Jun-2023].
- [31] tailwindcss. Responsive Design - Tailwind CSS — tailwindcss.com. <https://tailwindcss.com/docs/responsive-design>. [Accessed 29-Jun-2023].
- [32] Solidity Team. Solidity Programming Language — soliditylang.org. <https://soliditylang.org/>. [Accessed 29-Jun-2023].
- [33] trufflesuite. pet-shop. <https://trufflesuite.com/guides/pet-shop/>. [Accessed 29-Jun-2023].
- [34] trufflesuite. Truffle | Overview - Truffle Suite — trufflesuite.com. <https://trufflesuite.com/docs/truffle/>. [Accessed 29-Jun-2023].
- [35] typescriptlang.org. Documentation - The Basics — typescriptlang.org. <https://www.typescriptlang.org/docs/handbook/2/basic-types.html>. [Accessed 29-Jun-2023].

- [36] Vercel. Projects and deployments — vercel.com. <https://vercel.com/docs/getting-started-with-vercel/projects-deployments>. [Accessed 29-Jun-2023].
- [37] vitejs.dev. Vite — vitejs.dev. <https://vitejs.dev/guide/why.html>. [Accessed 29-Jun-2023].
- [38] Web3jsEthereum. web3.js - Ethereum JavaScript API x2014; web3.js 1.0.0 documentation — web3js.readthedocs.io. <https://web3js.readthedocs.io/en/v1.10.0/>. [Accessed 29-Jun-2023].
- [39] Wikipedia. Problema de los generales bizantinos — wikipedia, la enciclopedia libre, 2022. [Internet; Accessed 27-Jun-2023].
- [40] Wikipedia. Árbol de merkle — wikipedia, la enciclopedia libre, 2022. [Internet; descargado 26-julio-2023].
- [41] Wikipedia. Prueba de apuesta — wikipedia, la enciclopedia libre, 2023. [Internet; descargado 27-mayo-2023].
- [42] Wikipedia. Prueba de trabajo (algoritmo de consenso distribuido) — wikipedia, la enciclopedia libre, 2023. [Internet; Accessed 20-mayo-2023].
- [43] Wikipedia. React — wikipedia, la enciclopedia libre, 2023. [Internet; descargado 22-febrero-2023].
- [44] Wikipedia contributors. Hash function — Wikipedia, the free encyclopedia, 2023. [Online; accessed 27-June-2023].
- [45] Wikipedia contributors. Public-key cryptography — Wikipedia, the free encyclopedia, 2023. [Online; accessed 27-June-2023].
- [46] Dieter Shirley William Entriken (@fulldecent). Erc-721: Non-fungible token standard, Jan 2018.