
Trabajo de Fin de Máster

Máster Universitario en Ciberseguridad e Inteligencia de los Datos

# Sistema seguro basado en Blockchain para compartir vehículo autónomo

*Secure blockchain-based system for autonomous car sharing*

Néstor Torres Díaz

La Laguna, 6 de julio de 2024

D. **Cándido Caballero Gil**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Jezabel Miriam Molina Gil**, profesora Contratado Doctor de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

**C E R T I F I C A (N)**

Que la presente memoria titulada:

”*Sistema seguro basado en Blockchain para compartir vehículo autónomo*”

ha sido realizada bajo su dirección por D. **Néstor Torres Díaz**, con N.I.F. 54.064.450-Y.

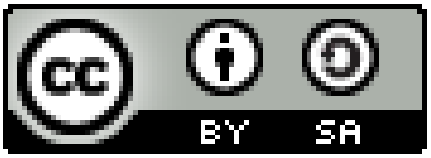
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2024

# Agradecimientos

Agradecer a mis padres, hermano y amigos, por siempre confiar en mis capacidades y animarme constantemente.

A mi tutor Cándido y a mi cotutora Jezabel por la oportunidad de investigar en este ámbito, por orientarme y ayudarme en la realización del trabajo.

# Licencia



**Resumen**

*La tecnología blockchain, concebida inicialmente para Bitcoin y la resolución del problema del doble gasto, ha evolucionado significativamente desde su origen. Un hito a destacar de esta evolución es el desarrollo de Ethereum, que no solo introdujo los contratos inteligentes, sino que también proporcionó una plataforma más flexible y multifuncional comparada con su predecesora, Bitcoin. Ethereum ha permitido que desarrolladores de todo el mundo creen aplicaciones descentralizadas (dApps) sobre su plataforma, expandiendo así las posibilidades de la tecnología blockchain más allá de las transacciones financieras. Esta innovación ha simplificado la automatización de procesos y la eliminación de intermediarios, mejorando la eficiencia y transparencia en varios sectores.*

*El objetivo de este trabajo, titulado "Sistema seguro basado en Blockchain para compartir vehículo autónomo", es crear una plataforma que emplee la blockchain para facilitar un servicio de alquiler de vehículos autónomos seguro y eficiente. Utilizando contratos inteligentes de Ethereum, el proyecto automatiza el alquiler desde la reserva hasta su conclusión, sin intermediarios, reduciendo costos operativos y mejorando tanto la seguridad de los datos como la confianza entre usuarios. Además, el sistema busca promover una movilidad sostenible, optimizando el uso de vehículos y contribuyendo significativamente a la reducción de la contaminación ambiental.*

**Palabras clave:** Blockchain, Ethereum, Contrato inteligente, Vehículos autonomos compartidos, Alquiler de vehículos

**Abstract**

*Blockchain technology, initially conceived for Bitcoin and the resolution of the double-spending problem, has evolved significantly since its origin. One milestone to note in this evolution is the development of Ethereum, which not only introduced smart contracts, but also provided a more flexible and multifunctional platform compared to its predecessor, Bitcoin. Ethereum has enabled developers around the world to create decentralized applications (dApps) on its platform, thus expanding the possibilities of blockchain technology beyond financial transactions. This innovation has simplified the automation of processes and the elimination of intermediaries, improving efficiency and transparency in various industries.*

*The goal of this work, titled "Secure blockchain-based system for autonomous car sharing", is to create a platform that employs blockchain to facilitate a secure and efficient autonomous vehicle rental service. Using Ethereum smart contracts, the project automates the rental from booking to completion, without intermediaries, reducing operational costs and improving both data security and trust between users. In addition, the system seeks to promote sustainable mobility, optimizing the use of vehicles and contributing significantly to the reduction of environmental pollution.*

**Keywords:** Blockchain, Ethereum, Smart contract, Shared autonomous vehicles, Car rental

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. Motivación

La tecnología Blockchain, conceptualizada inicialmente en el artículo *How to time-stamp a digital document (1), por Haber y Stornetta en 1991*, fue ideada para asegurar las marcas de tiempo de los documentos evitando su alteración o retrodatado. Su reconocimiento y desarrollo práctico despegó con la introducción de Bitcoin en el whitepaper *"A Peer-to-Peer Electronic Cash System" (2), por Satoshi Nakamoto en 2008*, que demostró el valor de esta tecnología en la creación de una moneda digital segura y descentralizada.

Desde entonces, la adopción de Blockchain ha avanzado enormemente, impulsada inicialmente por el éxito y la innovación de Bitcoin y otras criptomonedas. Un hito crucial fue la publicación del whitepaper de Ethereum *"A Next-Generation Smart Contract and Decentralized Application Platform" (3), por Vitalik Buterin en 2013*, que introdujo los contratos inteligentes y amplió las aplicaciones de la blockchain más allá de las finanzas hacia sectores como la logística, la sanidad y el gobierno digital. Los contratos inteligentes permiten gestionar aplicaciones de una forma eficiente y segura, donde su inmutabilidad garantiza que los términos del contrato no se pueden alterar una vez este ha sido desplegado en la Blockchain, asegurando la integridad y la inviolabilidad de las transacciones. Su transparencia permite que todas las partes involucradas puedan realizar auditorías y verificaciones continuas, y por otro lado, el hecho de que los contratos se ejecuten de manera automática y autónoma según los términos preestablecidos en los mismos, evitan la necesidad de intermediarios a la hora de realizar transacciones, abaratando y agilizando los procesos.

Teniendo en cuenta todas las ventajas que ofrece la Blockchain, se cree conveniente hacer uso de la misma para llevar el desarrollo de este Trabajo de Fin de Máster, siendo además una buena oportunidad para conocer más en profundidad esta tecnología.

## 1.2. Objetivos

El objetivo de este Trabajo de Fin de Máster es llevar a cabo el desarrollo de un **sistema seguro basado en Blockchain para compartir vehículo autónomo**, donde los usuarios puedan compartir sus vehículos autónomos con otros usuarios como una alternativa ecológica a los medios de transporte tradicionales.

Este concepto de compartir vehículos autónomos tiene la misma base y objetivo que otros servicios ya existentes como son el **"Scooter-Sharing"** o **"Bike-Sharing"**, que básicamente son servicios donde se comparte o alquila un medio de transporte por un corto período de tiempo. Este modelo ofrece numerosas ventajas al proporcionar una opción de transporte que es flexible y sostenible para el entorno, que reduce la dependencia de los vehículos personales y ayuda a disminuir tanto la congestión del tráfico como la contaminación del medio ambiente.

En concreto, se pretende aplicar la tecnología Blockchain para manejar el proceso de alquiler de los vehículos, permitiendo así llevar a cabo una gestión descentralizada, segura y transparente de dicho proceso.

## 1.3. Fases del desarrollo

Para proceder de manera organizada con el desarrollo, se definieron varias tareas distribuidas en distintas fases del proyecto, que se detallan a continuación:

- **Investigación y estado del arte del problema:** Se busca ampliar información sobre la tecnología Blockchain y su funcionamiento con el objetivo de adquirir los conocimientos básicos. Por otro lado, el estudio del arte del problema hace referencia a la investigación de aplicaciones, sistemas o plataformas que apliquen esta tecnología tanto en el ámbito de interés, como también en otros ámbitos.
- **Definición de requisitos:** Teniendo en cuenta la finalidad de la aplicación a desarrollar, se lleva a cabo un desglose de las características y funcionalidad mínimas que debe incorporar la aplicación para cumplir con el objetivo establecido para este trabajo.
- **Estudio de la adecuación de las tecnologías a utilizar:** Se planeó un estudio de las tecnologías para determinar las más adecuadas. Esta elección se basó principalmente en la facilidad de uso de las tecnologías, su integración con otras herramientas o lenguajes, el soporte y la documentación de las mismas, así como su popularidad dentro del desarrollo blockchain.
- **Desarrollo:** Se lleva a cabo el desarrollo de un sistema seguro para compartir vehículos autónomos haciendo uso de las tecnologías seleccionadas para cumplir con los objetivos previamente establecidos. Para ello se incluyen:
    - **Contrato inteligente:** Implementación de un contrato inteligente en Ethereum para gestionar los alquileres de vehículos, asegurando las transacciones de forma segura y sin intermediarios.

    - **Backend:** Una aplicación del lado del servidor para manejar la autenticación de usuarios y la información de usuarios y vehículos, utilizando una base de datos alternativa a la blockchain para almacenar los datos y para garantizar seguridad y eficiencia.
    - **Frontend:** Desarrollo de una interfaz de usuario accesible que se comunique con el backend y el contrato inteligente, facilitando a los usuarios el uso efectivo de la plataforma.

- **Redacción de la memoria:** Como tarea final, se recoge en la memoria todo el trabajo que se ha llevado a cabo durante este Trabajo de Fin de Máster, incluyendo el resultado final de la aplicación desarrollada.

## 1.4. Estructura de la memoria

Los subsiguientes apartados de la memoria se presentan acorde a la siguiente estructura:

- **Capítulo 2:** En este capítulo se introducen los conceptos básicos y fundamentales de la tecnología blockchain, así como su uso en la actualidad para el ámbito de compartir vehículos, de manera que se conozca el estado del arte actual del mismo.
- **Capítulo 3:** Además de indicar las tecnologías seleccionadas para el desarrollo de la aplicación, en este capítulo se muestran las funcionalidades implementadas y se comentan algunos detalles de la implementación.
- **Capítulos 4 y 5:** En ellos se resumen las conclusiones extraídas durante el desarrollo del trabajo de Fin de Máster (TFM) y se plantean posibles mejoras a futuro.
- **Capítulo 6:** En este último capítulo se muestran de manera desglosada los costos asociados al desarrollo del proyecto.

# Capítulo 2

# Antecedentes y Estado del Arte

Este capítulo comienza con una introducción a los conceptos fundamentales de la criptografía, esenciales para entender la tecnología blockchain. Se procede a explorar la estructura y el funcionamiento de la blockchain, incluyendo plataformas destacadas como Ethereum. Finalmente, se revisa el estado del arte, destacando una serie de aplicaciones que utilizan estas tecnologías para desarrollar sistemas o plataformas similares a la propuesta de este trabajo.

## 2.1. Criptografía

Las redes basadas en tecnología Blockchain, incluyendo Ethereum, utilizan la criptografía como pilar fundamental para el desarrollo y funcionamiento de sus plataformas. La criptografía permite la creación de sistemas de comunicación seguros que protegen la información frente a terceros no autorizados. Es mandatorio por lo tanto, llevar a cabo una introducción de varios conceptos criptográficos en los cuales se basa la tecnología Blockchain.

### 2.1.1. Funciones hash criptográficas

Una función hash es una función que recibe como entrada un mensaje de longitud arbitraria y devuelve un resumen o hash de longitud fija, que es lo que se conoce como el "hash" del mensaje.

Para que este tipo de funciones sean efectivas en un contexto donde la seguridad es esencial, como es el caso de la Blockchain, se exigen además una serie de propiedades que las funciones hash generales no necesariamente deben cumplir, pero las *funciones hash criptográficas* (4) si.

- **Determinismo:** El mismo mensaje siempre debe resultar en el mismo hash.
- **Eficiencia:** La función debe ser capaz de retornar el hash de un mensaje de manera rápida.
- **Resistencia a preimagen:** Debe ser computacionalmente inviable encontrar un mensaje que se corresponda con un hash dado.
- **Resistencia a segunda preimagen:** Debe ser computacionalmente inviable encontrar un segundo mensaje que tenga el mismo valor hash que un mensaje dado. Es

decir, dado un mensaje $m$ y su hash $h(m)$, debe ser muy difícil encontrar un segundo mensaje $m'$, tal que $h(m) = h(m')$.

- **Resistencia a colisiones:** Debe ser computacionalmente inviable encontrar cualquier par se mensajes diferentes $m$ y $m'$ que devuelvan como salida el mismo hash, es decir, $h(m) = h(m')$.

### 2.1.2. Criptografía de Clave Pública

La criptografía de clave pública, también conocida como criptografía asimétrica, es un sistema criptográfico que utiliza un par de claves para el envío de mensajes. Cada persona o entidad posee un par de claves, y estas pueden ser utilizadas para cifrar y descifrar mensajes así como también para firmarlos. La clave pública es utilizada para cifrar información o para verificar firmas digitales y puede ser compartida con cualquiera, mientras que la clave privada se utiliza para descifrar o firmar datos y tal como indica su nombre, esta no debe ser compartida con nadie.

El uso de la criptografía de clave pública para el envío de mensajes garantiza varios principios fundamentales de la seguridad de la información, los cuales juegan un papel muy importante en la protección de los datos:

- **Confidencialidad:** Cuando un usuario $A$ envía un mensaje a $B$ y lo cifra usando la clave pública de $B$, sólo la clave privada de $B$ puede descifrarlo. Esto garantiza que únicamente el destinatario intencionado tenga acceso al contenido del mensaje, manteniendo la confidencialidad de la información transmitida.

- **Autenticidad e Integridad:** Al enviar un mensaje, un usuario puede firmarlo usando su clave privada. Este proceso implica calcular el hash del mensaje y cifrarlo con su clave privada. El receptor, al recibir el mensaje, puede verificar esta firma utilizando la clave pública del emisor. Esta verificación asegura que el mensaje proviene realmente del emisor declarado (autenticidad) y que no ha sido alterado en tránsito (integridad).

- **No repudio:** El principio de no repudio también se cumple ya que el emisor no puede negar la autoría o el envío del mensaje. La firma digital creada con su clave privada, que es única y confidencial, verifica que él fue quien originó el mensaje.

### 2.1.3. Árbol de Merkle

El árbol de Merkle, ilustrado en la figura [2.1], es una estructura de datos fundamental en la tecnología blockchain para garantizar la integridad y la eficiencia de la verificación de los datos.

En esta estructura de datos cada nodo es etiquetado con el resultado de la función hash criptográfica de sus nodos hijo. Esto significa que a excepción de los nodos hoja, cada nodo es el resultado de la concatenación de los hashes de sus nodos hijos. Los nodos hoja, por su parte, son etiquetados directamente con el hash de los datos que representan. Este proceso de etiquetado continúa hasta alcanzar la raíz del árbol, la cual es un único hash que representa la integridad de todos los datos subyacentes.

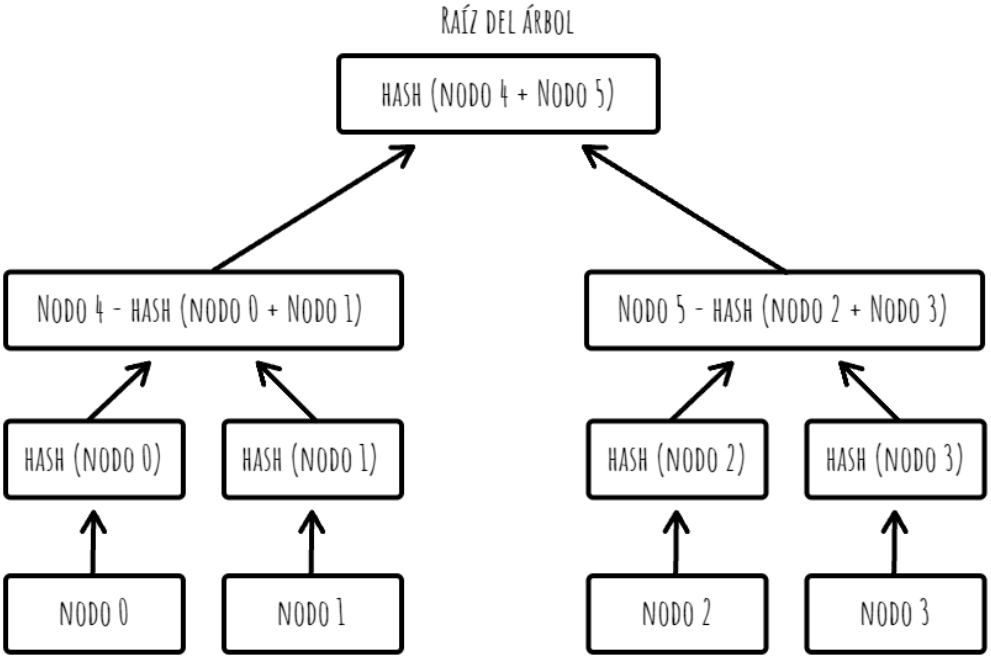



Figura 2.1: Ilustración del Árbol de Merkle

En blockchain, el árbol de Merkle juega un papel crucial al compactar y verificar grandes conjuntos de transacciones, asegurando su inalterabilidad. En el caso de Bitcoin, por ejemplo, cada bloque de la cadena incluye un hash de Merkle que resume todas las transacciones en ese bloque, no solo garantizando su integridad sino también permitiendo una rápida verificación de la inclusión de transacciones específicas a través de la "prueba de pertenencia de Merkle". Esto facilita la confirmación eficiente de que una transacción pertenece a un bloque sin necesidad de revisar toda la información contenida en él.

## 2.2. Blockchain

Blockchain es una tecnología de registro distribuido (Distributed Ledger Technology) que permite mantener una base de datos consensuada y segura sin la necesidad de una autoridad central. Tal y como se comentó en la introducción, los orígenes de esta tecnología se remontan a 1991 con la propuesta de Stuart Haber y W. Scott Stornetta para sellar documentos digitales y prevenir modificaciones en los mismos.

La primera aplicación práctica significativa de la Blockchain fue la creación de Bitcoin en el año 2008 por Satoshi Nakamoto (pseudónimo del creador o grupo de creadores de Bitcoin). En este caso se usó la Blockchain como el sistema subyacente para logar una moneda digital descentralizada y segura, sin necesidad de una tercera parte de confianza. Este sistema surgió como una alternativa a las monedas tradicionales e introdujo el concepto de minería como un método para alcanzar el consenso en una red distribuida mediante el mecanismo de Prueba de Trabajo (PoW).

### 2.2.1. Bloques

La cadena de bloques está compuesta por bloques tal y como indica su nombre, los cuales son estructuras de datos que registran las transacciones y se aseguran mediante criptografía. En concreto, cada bloque se compone de dos partes, una cabecera que contiene metadatos y una lista de transacciones que el bloque almacena. Un bloque típico en la cadena de bloques incluye:

- **Cabecera del bloque:** Contiene metadatos sobre el bloque, como el hash del bloque anterior, lo que enlaza de manera directa a cada bloque con su predecesor y forma una cadena continua e inmutable. La cabecera incluye también otros campos importantes como la raíz Merkle de las transacciones del bloque, el timestamp del momento en el que el bloque fue creado, el nivel de dificultad del algoritmo de consenso (Proof-of-work) y el nonce utilizado también por el algoritmo de consenso.
- **Lista de transacciones**: Detalla todas las transacciones validadas y confirmadas durante un período determinado, que se agregan al bloque.

En la figura [2.2] se ilustra la composición de una cadena de bloques.

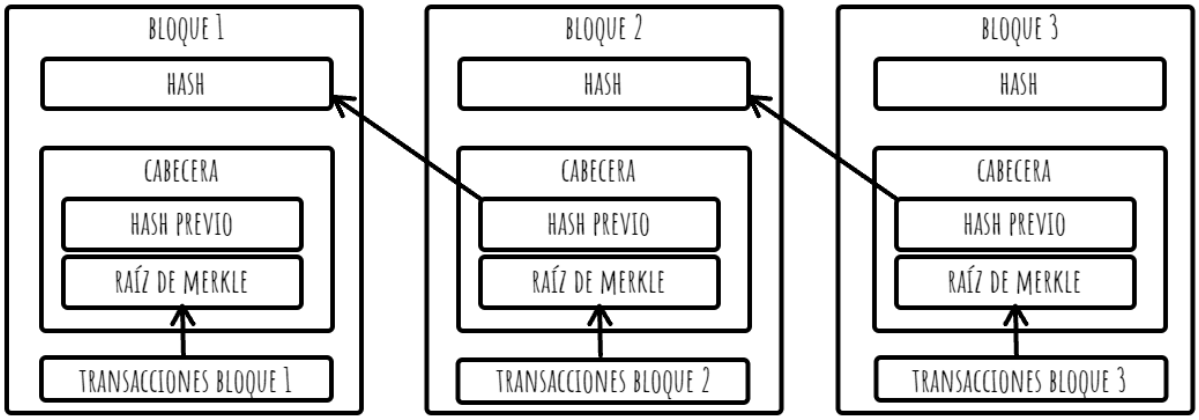



Figura 2.2: Ilustración de una cadena de bloques

El bloque génesis, siendo el primer bloque en cualquier cadena de bloques, no tiene un bloque anterior del cual derivar un hash, estableciendo su unicidad dentro de la cadena. Este bloque es generalmente configurado manualmente por los desarrolladores al iniciar una nueva red blockchain. Este acto no solo simboliza el inicio de la cadena, sino que también establece la base de seguridad y continuidad para todos los bloques que seguirán.

También es importante destacar que la inserción de un nuevo bloque en una blockchain que utiliza el mecanismo de consenso de prueba de trabajo (Proof of Work) requiere una considerable cantidad de poder computacional. Esto hace que la modificación de cualquier bloque, incluido el bloque génesis, sea prácticamente inviable debido al enorme esfuerzo computacional que implicaría recalibrar los hashes de todos los bloques subsiguientes.

### 2.2.2. Problema del consenso

En redes que operan sobre arquitecturas distribuidas Peer-to-peer (P2P), como las redes blockchain, se debe abordar el problema del consenso, que es la necesidad de que todos los nodos en la red acuerden el estado actual de la cadena para mantener la integridad y coherencia de los datos. Dado que la blockchain es descentralizada, sin una autoridad central reguladora que valide los cambios, los nodos mismos deben validar de forma consensuada las transacciones y bloques añadidos, aumentando así la seguridad y transparencia. Este problema se aborda mediante el uso de varios algoritmos de consenso diseñados para asegurar un acuerdo uniforme entre todos los nodos sin una autoridad central, entre los cuales destacan:

- **Prueba de Trabajo ó Proof of Work (PoW):** Los nodos compiten entre sí para resolver problemas matemáticos complejos y el primero en resolverlo gana el derecho de añadir un nuevo bloque a la cadena. Aunque es efectivo para mantener la seguridad de la red, requiere de una gran cantidad de energía y recursos computacionales.
- **Prueba de Participación ó Proof of Stake (PoS):** La probabilidad de que un nodo sea elegido para forjar un nuevo bloque es proporcional a la cantidad de monedas que este posee o tiene en "juego". Al no requerir de cálculos complejos se reduce signficativamente la cantidad de energía requerida, por lo que representa una alternativa más eficiente que el algoritmo PoW.
- **Prueba de Autoridad ó Proof of Authority (PoA):** Sólo un grupo preseleccionado de nodos tiene el derecho de llevar a cabo la validación de los bloques y transacciones, lo que hace que el proceso sea mucho más eficiente y menos demandante en términos de potencia de cálculo. Este algoritmo es una buena opción para redes privadas donde la identidad de los participantes es conocida y confiable.

En redes conocidas como Bitcoin se utiliza el algoritmo PoW desde sus inicios y permite la creación y validación de bloques a través del esfuerzo computacional de los mineros. Por su parte, Ethereum se inició haciendo uso de este mismo algoritmo de consenso, pero en el año 2022 transicionó a utilizar el algoritmo PoS, transición conocida como "The Merge" (5) la cual supuso una reducción del consumo de energía de la red de más del 99 %, marcando un hito importante hacia soluciones más sostenibles dentro de la tecnología blockchain.

### 2.2.3. Transacciones

Las transacciones representan acciones que modifican el estado de la cadena de bloques, ya sea mediante contratos inteligentes o transferencias de monedas. Estas representan un proceso crítico dentro de la blockchain y requieren del uso de claves criptográficas asociadas a cada usuario para garantizar tanto la seguridad como la autenticidad de las operaciones. El proceso que sigue una transacción en blockchain es el que sigue:

1. **Inicio de la transacción:** Un usuario $A$ (emisor) quiere enviar una cantidad $X$ de criptomonedas a otro usuario $B$ (receptor). Se recopila la información necesaria para la transacción, incluída la clave pública del receptor para asegurar que sólo el pueda acceder a los fondos.

2. **Firma de la transacción:** El emisor firma digitalmente la transacción haciendo uso de su clave privada. Esto se realiza a través de una función de firma que toma como entrada los datos de la transacción y la clave privada del emisors, obteniendo la firma digital como resultado. Esta firma, junto a la clave pública del emisor y los datos de la transacción, se transmiten a la red.

3. **Verificación de la transacción:** Los nodos de la red, incluyendo los mineros y el receptor, reciben y verifican la transacción, asegurandose de que la persona que envía la transacción es quien dice ser y no ha sido modificada, además de comprobar que el emisor posee el saldo suficiente como para llevar a cabo esa transacción, evitando el problema del doble gasto. Este problema surge porque en el contexto de las criptomonedas y otras formas de dinero digital, existe la posibilidad de que una misma unidad de moneda digital pueda ser gastada más de una vez, lo cual no ocurre con el dinero físico porque la entregar una moneda o billete como pago, ya no se posee y por lo tanto no se puede volver a utilizar.

4. **Integración de la transacción en el bloque:** Una vez se ha verificado la transacción, esta se incluye junto a otras transacciones válidas en un nuevo bloque que está en proceso de construcción y que se prepara para ser añadido a la cadena de bloques.

5. **Integración del bloque en la cadena de bloques:** Cuando el bloque está completo, se distribuye a toda la red para que sea validado y solo se acepta si todas las transacciones dentro del bloque son válidas. Los nodos que validan el bloque lo añaden a su copia local de la blockchain y esto ocurre de manera sucesiva con los nuevos bloques que se van construyendo.

6. **Finalización de la transacción:** Una vez que el bloque con la transacción es aceptado y añadido a la cadena de bloques, la transacción se considera como ejecutada. En este punto, la transferencia de criptomonedas se ha llevado a cabo y el receptor ya tiene como disponibles los fondos recibidos.

En la figura [2.3] se ilustra el proceso que sigue una transacción en la blockchain.

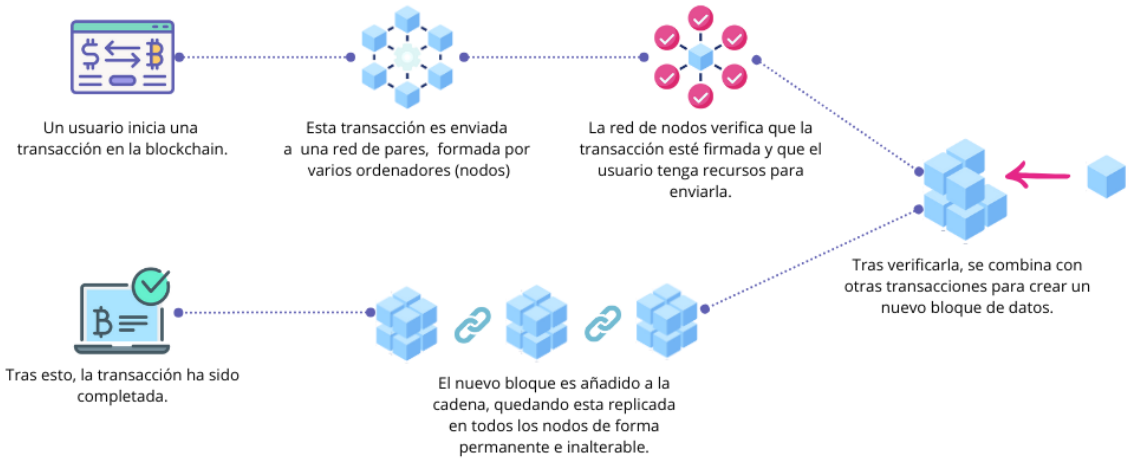



Figura 2.3: Ilustración simplificada de una transacción en la blockchain

### 2.2.4. Tipos de blockchain

En el ámbito de la tecnología blockchain, es posible clasificar las redes según su estructura de acceso y control. Esta clasificación es crucial para entender cómo cada tipo de blockchain se implementa y opera, variando en función de sus objetivos específicos y contextos de aplicación.

- **Blockchain Pública:** Es completamente abierta y cualquier persona con conexión a Interned puede unirse a la red como usuario o como nodo validador (minero). Esto promueve un alto nivel de transparencia teniendo en cuenta que todas las transacciones son públicamente accesibles y verificables por cualquiera de la red.
- **Blockchain Privada o Permisionada:** Contraria a la blockchain pública, la blockchain privada restringe el acceso a redes específicamente diseñadas para entornos corporativos o cerrados, por lo que la participación en la misma está limitada a entidades autorizadas y esto permite tener un control centralizado sobre quíen puede interactuar con la blockchain.
- **Blockchain Híbrida:** Busca equilibrar la transparencia de las redes públicas con el control de las redes privadas. Este tipo de blockchain permite operar en un entorno controlado mientras se hace accesible parte de la información al público o a un grupo más amplio de partes interesadas sin comprometer la seguridad de los datos sensibles. Esto es ideal para organizaciones que necesitan compartir información de una manera selectiva y fuera de sus límites corporativos pero sin sacrificar la privacidad y la seguridad interna.

## 2.3. Ethereum

La plataforma de Ethereum [2.4] es una de las tecnologías de blockchain más influyentes y ampliamente adoptadas después de Bitcoin. Ethereum fue propuesta en 2013 por Vitalik Buterin y desarrollada por Buterin y Gavin Wood, siendo lanzada en el año 2015. A diferencia de Bitcoin, que se limita principalmente a operaciones de monedas digitales, Ethereum se construye como una plataforma para ejecutar contratos inteligentes y aplicaciones descentralizadas (DApps), utilizando su criptomoneda nativa, el ether (ETH).

Figura 2.4: Logo de la plataforma Ethereum

El ether no solo sirve como medio de intercambio o como activo, sino que también es crucial para operar y desarrollar en la red de Ethereum, ya que se utiliza para pagar las tarifas asociadas a las transacciones y los servicios computacionales, facilitando de esta manera la ejecución de los contratos inteligentes y el funcionamiento de las DApps. El ether puede ser expresado también en otras unidades más pequeñas, las cuales son llamadas las “denominaciones del ether”, ilustradas en la tabla [2.1].

| Nombre de la Unidad | Valor en Wei | Cantidad de Wei |
|---|---|---|
| Wei (wei) | 1 wei | 1 |
| Kwei (babbage) | $10^3$ wei | $1,000$ |
| Mwei (lovelace) | $10^6$ wei | $1,000,000$ |
| Gwei (shannon) | $10^9$ wei | $1,000,000,000$ |
| Twei (szabo) | $10^{12}$ wei | $1,000,000,000,000$ |
| Pwei (finney) | $10^{15}$ wei | $1,000,000,000,000,000$ |
| Ether (buterin) | $10^{18}$ wei | $1,000,000,000,000,000,000$ |

Tabla 2.1: Denominaciones del Ether

### 2.3.1. Contratos inteligentes

El concepto de contrato inteligente fue introducido por Nick Szabo mucho antes de la creación de Ethereum, en 1996 a través del whitepaper *"Smart Contracts: Building Blocks for Digital Markets" (6)*, como una forma de extender la funcionalidad de los contratos electrónicos tradicionales mediante la incorporación de protocolos informáticos para facilitar, verificar o imponer el cumplimiento de un contrato. Con la llegada de Ethereum, se pudo materializar este concepto gracias a que la plataforma proporcionó el entorno necesario para implementar estos contratos de una manera segura y descentralizada.

Los contratos inteligentes en Ethereum son programas que se ejecutan exactamente como fueron programados sin posibilidad de fraude, censura o interferencia de terceros. Estos contratos se implementan utilizando lenguajes de programación de alto nivel, como Solidity.

### 2.3.2. Máquina Virtual de Ethereum

La Máquina Virtual de Ethereum o Ethereum Virtual Machine (EVM) es el entorno de ejecución para los contratos inteligentes en Ethereum, cuyo funcionamiento se ilustra en la figura [2.5].

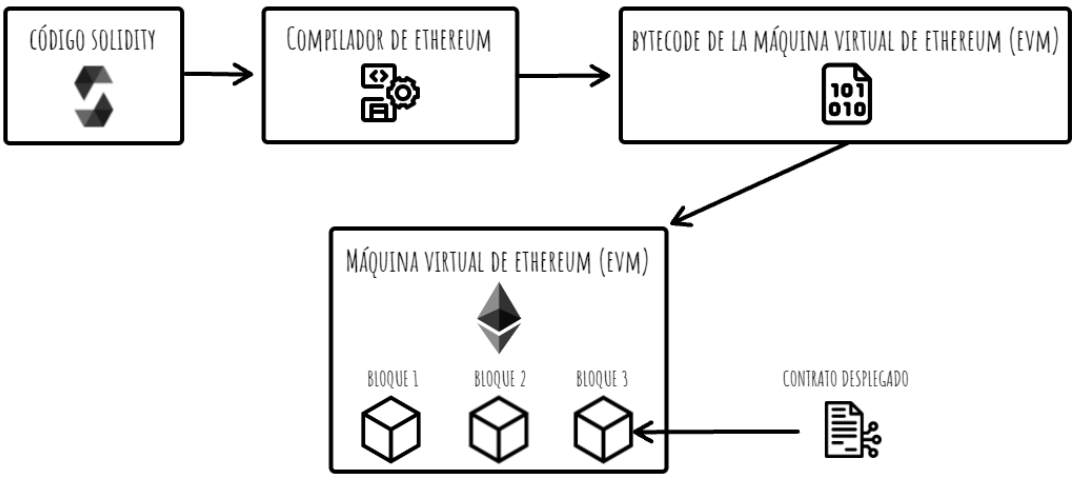



Figura 2.5: Esquema simplificado del funcionamiento de la Máquina Virtual de Ethereum (EVM)

Funciona como una plataforma descentralizada que soporta el código de los contratos inteligentes, permitiendo su ejecución con un alto grado de seguridad y aislamiento. Completamente aislada del resto de la red y sin acceso al sistema de archivos o a otros recursos del sistema del nodo que ejecuta Ethereum, la EVM asegura que los contratos solo puedan interactuar entre ellos de manera controlada y predecible.

Cada nodo en la red de Ethereum contiene una copia de la EVM, y cada uno de ellos ejecuta el mismo contrato en paralelo, por lo que cada transacción que se lleve a cabo en un contrato inteligente es procesada y verificada por cada nodo, asegurando la integridad y la inmutabilidad de la red.

Además, la EVM permite la compatibilidad entre versiones de la cadena de bloques y la portabilidad del código, facilitando la creación de aplicaciones que no están asociadas a ningún tipo particular de blockchain. Esto hace a Ethereum una plataforma atractiva para el desarrollo de aplicaciones descentralizadas que requieren un entorno de ejecución robusto y seguro.

### 2.3.3. Gas

La Máquina Virtual de Ethereum (EVM) utiliza una unidad denominada “gas” para cuantificar el esfuerzo computacional que requieren las operaciones. Cada operación ejecutable por un contrato inteligente, desde simples cálculos hasta funciones más complejas, tiene asignado un costo específico en gas que depende de la complejidad de la operación, de manera que si una operación demanda una ingente cantidad de recursos de computación, su coste de gas será muy elevado, lo que ayuda a prevenir comportamientos perjudiciales como bucles infinitos que podrían dañar la red.

Cuando se realiza una transacción, el usuario debe definir una “oferta de gas”, que es el precio que el usuario está dispuesto a a pagar por cada unidad de gas, expresado en ether. Además, se debe especificar un “límite de gas”, que se corresponde con la cantidad máxima de gas que el usuario está dispuesto a pagar por la transacción. El establecer un límite de gas adecuado permite que el usuario no pague más de lo necesario y le protege de ejecuciones excesivamente costosas en caso de errores en el contrato.

Todos estos mecanismos de control del gas aseguran que los usuarios solo paguen por los recursos utilizados y ofrecen una forma eficiente para limitar y gestionar el costo de ejecuciones tanto de transacciones como de contratos inteligentes en la red Ethereum.

### 2.3.4. Cuentas

Ethereum distingue entre dos tipos principales de cuentas, cada una con roles y características específicas dentro de la red:

- **Cuentas de usuario (Externally Owned Accounts - EOAs):** Estas cuentas están controladas por claves privadas y no tienen código asociado. Los usuarios suelen hacer uso de estas cuentas para enviar transacciones, que pueden incluir transferencias de ether y la interacción con contratos inteligentes en la red. Para que una transacción sea válida, tal y como comentamos en apartados anteriores, esta debe

estar firmada digitalmente haciendo uso de la clave privada asociada a la cuenta del usuario, garantizando la autenticidad, la integridad y la autorización del proceso.

- **Cuentas de contrato (Contract Accounts - CAs):** A diferencia de las cuentas de usuario, las cuentas de contrato contienen código que se ejecuta cuando reciben transacciones. Estos contratos inteligentes son capaces de definir reglas, almacenar datos y automáticamente ejecutar lo establecido dentro del contrato según las interacciones que este reciba. El código de un contrato puede enviar transacciones, crear nuevos contratos e interactuar con otros contratos, facilitando la creación de aplicaciones descentralizadas complejas.

A continuación, se presenta en [2.6] un esquema explicativo sobre la estructura de los tipos de cuentas de la plataforma Ethereum.

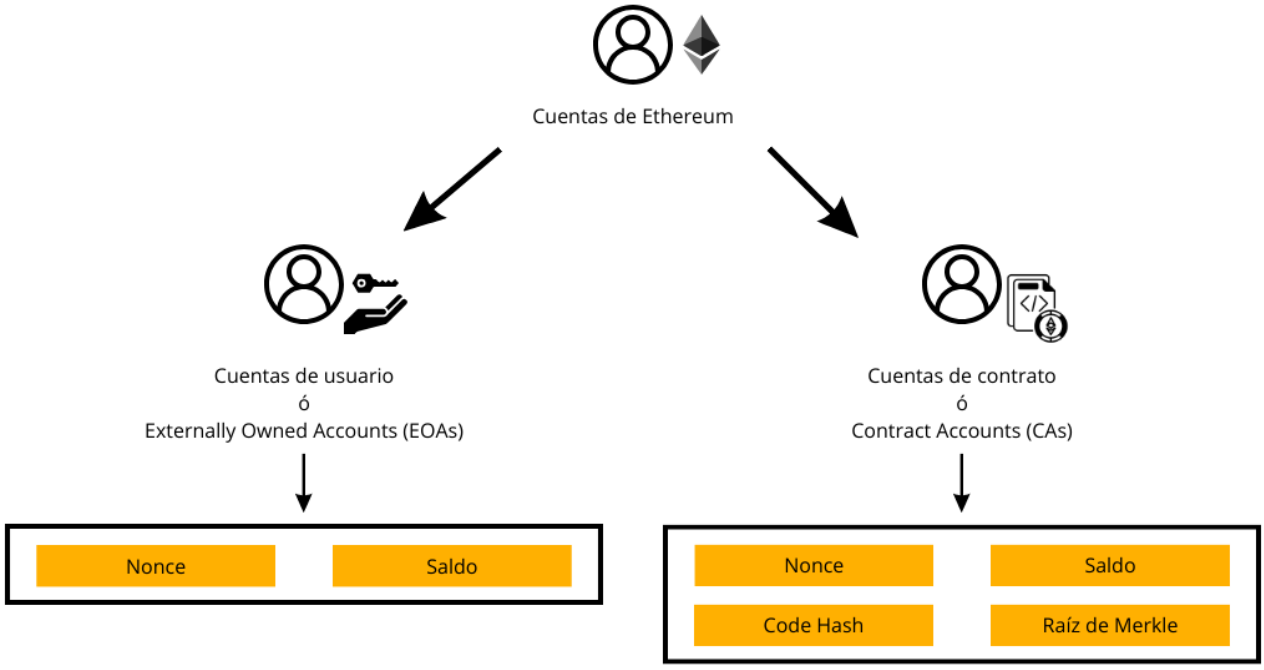



Figura 2.6: Esquema simplificado de los tipos de cuentas de Ethereum

Cabe destacar, que cada cuenta en Ethereum está identificada por una dirección única de 20 bytes. Esta dirección, que comienza con $0x$, actúa como un identificador público para la recepción de fondos o la interacción dentro de la red.

## 2.4. Aplicaciones

En este apartado se van a mencionar algunas aplicaciones que, alineadas con el objetivo propuesto para el desarrollo de este Trabajo de Fin de Máster, hacen también uso de la tecnología blockchain para implementar sistemas de alquiler de vehículos o similares.

### 2.4.1. DAV

Plataforma descentralizada diseñada para integrar vehículos autónomos, incluidos coches, camiones, drones y otros, en una red que permite descubrir, comunicarse y transaccionar sin intermediarios. Lanzada en junio de 2017 y ubicada en Suiza, DAV (7) introduce un modelo económico basado en tokens que facilita el acceso a la red. Utiliza contratos inteligentes para gestionar las interacciones dentro de la red, asegurando transacciones transparentes y eficientes. Además, DAV forma parte de la iniciativa MOBI, colaborando con grandes fabricantes de automóviles y otras entidades para promover el uso de tecnologías de transporte descentralizadas. Su enfoque no solo se centra en la facilitación de transporte autónomo, sino también en la creación de un ecosistema económico viable y de amplio alcance para todos los participantes de la red.

Figura 2.7: Logo de la plataforma DAV

### 2.4.2. Helbiz

Plataforma descentralizada que se centra en la micro-movilidad global, ofreciendo una flota diversa de vehículos como e-scooters, e-bicicletas y e-mopeds a través de una plataforma conveniente y fácil de usar. Fundada en 2015 y con sede en Nueva York, Helbiz (8) ha expandido su presencia globalmente, operando en varias ciudades alrededor del mundo. A diferencia de muchas otras plataformas, Helbiz se distingue por utilizar su propia criptomoneda, HelbizCoin, para facilitar las transacciones dentro de su ecosistema. Esta criptomoneda ya está en circulación y se integra en las transacciones para ofrecer una experiencia de usuario fluida y segura. Además de esto, la compañía ha implementado tecnología avanzada de gestión de flotas, inteligencia artificial y mapeo ambiental para optimizar sus operaciones y la sostenibilidad del negocio.


Figura 2.8: Logo de la plataforma Helbiz

# Capítulo 3

# Desarrollo de la aplicación

La aplicación desarrollada en este Trabajo de Fin de Máster, se llama **SoloDrive** y el código asociado a la misma se encuentra alojado en un repositorio público (9). En este capítulo se mencionarán las tecnologías utilizadas, las funcionalidades de la aplicación y también ciertos detalles sobre los contratos inteligentes implementados.

## 3.1. Tecnologías utilizadas

### 3.1.1. Vue.js y Vuetify

Se ha seleccionado el framework de Javascript Vue.js (10) para llevar a cabo el desarrollo de la aplicación web. Este framework permite crear aplicaciones de una sola página y también interfaces de usuario. Entre las principales razones que motivan esta elección se encuentran la facilidad de uso, la flexibilidad que dispone para reemplazar componentes fácilmente, la modularidad y la reactividad ante los cambios sin necesidad de recargar la página. En concreto, la flexibilidad de Vue.js permite que se pueda hacer uso de bibliotecas de componentes como Vuetify (11), las cuales proveen de componentes completos que permiten agilizar el desarrollo de las interfaces de usuario.

Figura 3.1: Logos de Vue.js y Vuetify

### 3.1.2. MongoDB y NestJS

Dada la necesidad de disponer de un sistema de persistencia de datos alternativo a la blockchain para manejar de manera efectiva la información de usuarios y vehículos, se ha optado por MongoDB (12). Esta base de datos no relacional, orientada a documentos, destaca por su compatibilidad con numerosos lenguajes de programación, su capacidad para ejecutar consultas ad hoc, y su almacenamiento de datos en documentos flexibles con un formato similar a JSON, lo que hace que su uso sea fácil e intuitivo. Además,

MongoDB tiene también la capacidad de realizar consultas utilizando Javascript (13), las cuales son enviadas directamente a la base de datos para ser ejecutadas.

Figura 3.2: Logo de MongoDB

Para añadir una capa de seguridad a la aplicación, se requiere un sistema de autenticación para usuarios en el lado del servidor, aprovechando la información almacenada en MongoDB. Para este propósito, se ha seleccionado seleccionado NestJS (14), un framework progresivo basado en Node.js (15) y desarrollado en Typescript (16), el cual fue diseñado para desarrollar aplicaciones backend robustas y escalables. NestJS se distingue por ofrecer un alto nivel de abstracción que simplifica de manera significativa el desarrollo de aplicaciones, facilitando la integración con diversas tecnologías. En concreto, NestJS se integra de manera eficiente con MongoDB gracias a su módulo integrado **@nestjs/mongoose** (17). Este módulo encapsula el driver de Mongoose, proporcionando una interfaz simplificada para modelar datos y manejar conexiones de base de datos.

Figura 3.3: Logo de NestJS

### 3.1.3. Solidity

Para la creación de contratos inteligentes que operan en la red de Ethereum, se ha seleccionado el lenguaje de programación Solidity (18), el cual es orientado a objetos. Este lenguaje es considerado estándar de facto para el desarrollo de contratos inteligentes en Ethereum y destaca por su gran compatibilidad con herramientas de desarrollo líderes en el sector, como Truffle, Hardhat y Remix, además de disponer de una gran comunidad que lo respalda, una extensa documentación y actualizaciones constantes para mejorar seguridad.

Figura 3.4: Logo de Solidity

### 3.1.4. Truffle y Ganache

Truffle (19) es un entorno de desarrollo integral, marco de pruebas y sistema de gestión de activos para blockchain que utiliza la Máquina Virtual de Ethereum (EVM), enfocado en el desarrollo, testeo y despligue de contratos inteligentes en la red Ethereum. Forma parte de la Truffle Suite (20), un conjunto de herramientas integradas que han sido diseñadas para el desarrollo de aplicaciones descentralizadas (DApps) sobre blockchains compatibles con la EVM.

Aparte de seleccionar Truffle como entorno de desarrollo de la aplicación a implementar, se selecciona también otra herramienta de la Truffle Suite, que en este caso es Ganache (21), una blockchain privada que simula el comportamiento de una red pública de Ethereum, de manera que se puedan realizar pruebas y desarrollar contratos inteligentes sin necesidad de gastar ether real.

Figura 3.5: Logos de Truffle y Ganache

### 3.1.5. Web3

Para interactuar con la blockchain, es esencial tener acceso a un nodo de la red. Esto se logra a través del protocolo Remote Procedure Call (RPC), que permite a un programa ejecutar procedimientos en un espacio de direcciones remoto como si fueran locales. En concreto, para llevar a cabo la comunicación con el nodo de Ethereum, se hará uso de la biblioteca Web3.js (22), la cual facilita la realización de llamadas JSON-RPC desde una aplicación web.

Figura 3.6: Logo de Web3.js

### 3.1.6. Metamask

Metamask (23) es una extension de navegador esencial para interactuar con la blockchain de Ethereum, actuando como una cartera digital que permite a los usuarios almacenar y gestionar sus cuenta de Ethereum con alta seguridad. Además, facilita la firma de transacciones y mensajes sin revelar las claves privadas, lo que refuerza la seguridad y mejora la experiencia de usuario al interacturar con las DApps. Por otro lado, Metamask ofrece una interfaz intuitiva que permite a los usuarios alternar fácilmente entre diversas redes blockchain, como testnests, privatenets o la mainnet, así como también entre diferentes wallets.

Figura 3.7: Logo de Metamask

## 3.2. Base de datos

Dado que tanto los usuarios de la plataforma como los vehículos representan información dinámica que se actualiza de manera constante, gestionar todos esos datos directamente en la blockchain resultaría en un elevado costo de gas. Este gasto viene asociado a las operaciones de escritura en la blockchain, como por ejemplo añadir, modificar o eliminar vehículos. Por lo tanto, se ha decidido almacenar la información de usuarios y vehículos exclusivamente en una base de datos MongoDB, tal y como se comentó en el apartado de tecnologías utilizadas.

Para llevar a cabo el modelado de datos, como ya se mencionó anteriormente, se utiliza **Mongoose**, una biblioteca de modelado de objetos para Node.js que actúa como una interfaz directa con MongoDB. Funciona como un ODM (Object Data Modelling), que facilita la definición de esquemas para los datos que se almacenan en MongoDB, proporcionando una estructura de datos clara y fácil de entender. En este caso concreto, se hace uso del paquete dedicado de Mongoose para NestJS, por lo que la definición de los modelos y la comunicación con la base de datos es distinta, ya que se hace uso de los decoradores de NestJS para crear los schemas, haciendo que mejore mucho la legibilidad del código.

```
@Schema()
export class User {
    @Prop({ required: true })
    username: string

    @Prop({ required: true, unique: true })
    dni: string

    @Prop({ required: true })
    password: string

    @Prop()
    wallet: string
}

export type UserDocument = User & Document

export const UserSchema = SchemaFactory.createForClass(User)
```

Figura 3.8: Modelo de usuario - MongoDB + Mongoose + NestJS

En este caso, podemos ver en [3.8] como para los usuarios de la plataforma se almacenan en el modelo un nombre de usuario, su documento de identidad, una contraseña (la cual es hasheada antes de almacenarla) y la dirección de su wallet de ethereum.

```
@Schema()
export class Car {
    @Prop({ required: true })
    model: string

    @Prop({ required: true, unique: true })
    plate: string

    @Prop({ required: true })
    autonomy: number

    @Prop({ required: true })
    price: number

    @Prop({ required: true })
    owner: string

    @Prop({ required: true })
    imageUrl: string

    @Prop({ default: true })
    available: boolean
}

export type CarDocument = Car & Document

export const CarSchema = SchemaFactory.createForClass(Car)
```

Figura 3.9: Modelo de vehículo - MongoDB + Mongoose + NestJS

Por otro lado, en [3.9] vemos que para los modelos de los vehículos se contemplan datos como el modelo del vehículo, la matrícula, la autonomía en kilómetros, el precio diario en dólares, la dirección de la wallet del propietario del vehículo, la dirección del enlace a la imagen asociada al vehículo y por último, la disponibilidad.

## 3.3. Contrato inteligente

Para gestionar la parte más delicada de la aplicación se plantea el contrato inteligente `CarRental`, el cual permite gestionar los alquileres de los vehículos, las devoluciones de los mismos y las transacciones asociadas a estas operaciones.

En este contrato, cuando se confirma el alquiler o renta de un vehículo se almacena información que permite hacer una asociación entre el vehículo, el cliente que lo alquila y el propietario que está compartiendo su vehículo. De esta manera, se puede controlar qué vehículos están alquilados o no y también se puede manejar de manera cómoda el pago del alquiler, ya que se conoce tanto la dirección de la wallet de ethereum del cliente como la del propietario. En concreto, la información almacenada en la blockchain sobre la renta de un vehículo es la que es muestra en [3.10].

```
struct Rental {
    string plate;
    uint price;
    string contactPhone;
    address client;
    address owner;
    bool active;
}
```

Figura 3.10: Estructura de la información sobre el alquiler del vehículo

### 3.3.1. Compilación, migración y despliegue

Para poder compilar el contrato implementado en Solidity, se hace uso del compilador de Truffle y para ello es necesario que el contrato a compilar tenga la extensión correspondiente de un archivo Solidity **(.sol)** y además que esté dentro del directorio `contracts/`. Una vez realizado todo esto, se puede proceder con la compilación del contrato ejecutando el comando “truffle compile”. Una vez compilado el contrato inteligente, se genera un fichero en formato JSON en el directorio `build/contracts`, el cual será utilizado por Truffle cuando se vaya a desplegar el contrato y también es necesario para que bibliotecas como Web3.js permitan interactuar con el contrato desde la aplicación web.

Para el despliegue de un contrato inteligente en la red, se puede hacer uso de:

- **Redes principales ó Mainnets** (Ethereum Mainnet en este caso)
- **Redes de prueba ó Testnets** (Ropsten, Rinkeby, Goerli, Kovan)
- **Redes privadas ó Privatenets** (Ganache, Hyperledger Besu, Quorum)

Tanto las redes de prueba como las privadas no incurren en gastos reales, por lo que en este caso se ha optado por escoger Ganache para montar una red privada que contiene la blockchain local donde se va a desplegar el contrato. Para poder especificar al framework Truffle donde se ubica esta red, se plantea un fichero de configuración [3.11], donde se concretan la dirección IP privada de la red local, el puerto en el que está escuchando y por último, el identificador de red.

```
module.exports = {
  networks: {
    development: {
      // host: "127.0.0.1",
      host: "172.31.64.1",
      port: 7545,
      network_id: "5777"
    }
  },
  compilers: {
    solc: {
      version: "0.8.19"
    }
  }
};
```

Figura 3.11: Configuración del framework Truffle

Por último, se debe especificar la migración asociada al contrato para que este puede ser desplegado en la red configurada. Esta migración se especifica en [3.12] a través un fichero Javascript de configuración. Tras esto, se ejecuta el comando “truffle migrate” para desplegarlo. En Truffle, una migración hace referencia al proceso de despliegue y gestión de contratos inteligentes en una red de blockchain.

```
const CarRental = artifacts.require("CarRental");

module.exports = async function(deployer) {
  await deployer.deploy(CarRental)
};
```

Figura 3.12: Script de Migración de Truffle

### 3.3.2. Funciones principales

Aunque ya se introdujeron con anterioridad las funciones principales del contrato, aquí se muestran los detalles de la implementación de dichas funciones.

#### Alquilar vehículo

En primera instancia, para manejar la acción de alquilar un coche se plantea la función `rentCar`, la cual recibe los siguientes parámetros:

- Matrícula del vehículo
- Precio que tiene alquilar el vehículo por un día
- Teléfono de contacto
- Dirección de la wallet del propietario del vehículo

La primera comprobación que se realiza es que el vehículo no esté alquilado ya a otro usuario. En caso de que no esté alquilado, se almacenan los datos comentados con anterioridad en combinación con la dirección de la wallet del cliente, el cual es la persona que invoca al contrato. Tras esto se lleva a cabo el pago del alquiler desde la wallet del cliente a la del propietario. En este caso, vemos en [3.13] que la cantidad enviada al propietario no es la misma que `price`, sino que es la especificada en el objeto `msg` y esto se debe a que en la aplicación web se lleva a cabo la conversión de dólares a ETH para que la cantidad de ethers enviada al propietario sea la correcta.

```
function rentCar(string memory _plate, uint _price, string memory _contactPhone, address _owner) public payable {
    require(activeRentals[_plate].client == address(0), "Car is already rented");

    Rental memory newRental = Rental({
        plate: _plate,
        price: _price,
        contactPhone: _contactPhone,
        client: msg.sender,
        owner: _owner,
        active: true
    });

    activeRentals[_plate] = newRental;
    activePlates.push(_plate);
    rentalHistory.push(newRental);

    // Transferimos pago al propietario
    payable(_owner).transfer(msg.value);
}
```

Figura 3.13: CarRental Contract - Función para alquilar vehículo

Podemos observar en [3.14] como tras instanciar el contrato inteligente, se lleva a cabo dicha conversión para posteriormente transferir la cantidad de ethers correspondiente desde la wallet del cliente hasta la del propietario del vehículo.

```
async function rentCar(plate, price, contactPhone, owner) {

  try {
    // Obtenemos la wallet del cliente
    const client = await getClientAccount()

    // Creamos la instancia del contrato
    const contract  = new window.web3.eth.Contract(carRentalABI,  carRentalAddress)

    const amountInEther = await convertUSDtoETH(price)

    await contract.methods.rentCar(plate, price, contactPhone, owner)
    .send({ from: client, value: window.web3.utils.toWei(amountInEther.toString(), 'ether')  })

    console.log('Rental contract created successfully')

    await API.patch(`cars/${plate}`)
    console.log('Car availability updated succesfully in API')

    router.push('/cars/available')
  } catch (error) {
    console.error('Rental contract failed: ', error)
  }
}
```

Figura 3.14: Aplicación web (Vue) - Función para alquilar vehículo

Cabe destacar en este caso, que además de almacenar la renta dentro de un mapping de Solidity relativo a las rentas activas de vehículos, también se almacena dicha renta en un array a modo de historial de rentas. Esto lo que permite es tener una mejor trazabilidad de todas las rentas que se lleven a cabo, lo cual podría utilizarse también para mostrar a los usuarios información histórica sobre su actividad en la plataforma.

### Devolver vehículo

Para llevar a cabo la devolución del vehículo, se plantea la función `returnCar`, ilustrada en [3.15], la cual únicamente recibe como parámetro la matrícula del vehículo a devolver. Lo primero que se lleva a cabo es la comprobación de que el vehículo esté alquilado, porque de lo contrario no tendría sentido realizar la devolución del mismo. Tras esto, se almacena en el historial de rentas un registro de asociado a la devolución del vehículo y luego se elimina esa renta del mapeo de rentas activas y de la lista de matrículas activas.

```
function returnCar(string memory _plate) public onlyClient(_plate) {
    Rental storage rental = activeRentals[_plate];
    require(rental.active, "Car is not currently rented");

    rentalHistory.push(Rental({
        plate: rental.plate,
        price: rental.price,
        contactPhone: rental.contactPhone,
        client: rental.client,
        owner: rental.owner,
        active: false
    }));

    // Eliminamos la matrícula del array de matrículas activas
    for (uint i = 0; i < activePlates.length; i++) {
        if (keccak256(abi.encodePacked(activePlates[i])) == keccak256(abi.encodePacked(_plate))) {
            activePlates[i] = activePlates[activePlates.length - 1];
            activePlates.pop();
            break;
        }
    }

    // Eliminamos la renta (por lo que no hace falta actualizar su estado)
    delete activeRentals[_plate];
}
```

Figura 3.15: CarRental Contract - Función para devolver vehículo

Se debe tener en cuenta que cuando un vehículo es devuelto, se debe actualizar su estado a disponible, pero teniendo en cuenta que la información de los vehículos se encuentra ubicada en la base de datos, debemos actualizar esto a través de la API implementada con NestJS, la cual es invocada desde la aplicación web tal y como se muestra en [3.16].

```
async function returnCar(plate) {
    const client = await getClientAccount()

    try {
    // Creamos la instancia del contrato
    const contract  = new window.web3.eth.Contract(carRentalABI,  carRentalAddress)

    await contract.methods.returnCar(plate).send({ from: client })
    console.log('Rental contract deleted successfully')

    await API.patch(`cars/${plate}`)
    console.log('Car availability updated succesfully in API')

    await getClientCars()
    } catch (error) {
        console.error('Failed to return car: ', error)
    }
}
```

Figura 3.16: Aplicación Web (Vue) - Función para devolver vehículo

### 3.3.3. Transacciones

Una transacción en Ethereum es una operación que se envía de una cuenta a otra y puede tratarse de una llamada a un contrato inteligente que acaba modificando el estado de la blockchain o incluso de una transferencia de ether.

Teniendo esto en cuenta, si observamos cómo se invoca el método `returnCar` del contrato inteligente en la aplicación web [3.16], vemos que junto con el método `send` no se especifica ninguna cantidad de Ether, por lo que no se trata de una transferencia de Ether. Sin embargo, debe ser invocado con el método `send` porque al ser una transacción que modifica el estado de la blockchain, esa operación requiere gas y, por lo tanto, incurre en un costo de Ether.

Por otro lado, tenemos la invocación del método `rentCar` del contrato inteligente [3.16], el cual también se invoca con el método `send` pero en este caso si que se especifica una cantidad concreta de Ether, por lo que en este caso esta transacción sí que se corresponde con una transferencia de ether.

Por último, destacar que cuando se invocan métodos de un contrato inteligente haciendo uso del método `.call()` en vez del método `.send()`, lo que se está llevando a cabo no es una transacción, sino que se está realizando una operación de lectura la cual no modifica el estado de la blockchain, tal y como podemos observar en [3.17].

```
// Obtenemos los coches alquilados por el cliente
const clientActiveRentals = await contract.methods.getActiveRentalsByClient(client).call()
```

Figura 3.17: Función del contrato inteligente para obtener las rentas activas de un cliente

## 3.4. Funcionalidades de la aplicación

Al acceder a la plataforma web, se muestra la página principal de la misma o landing page [3.18], donde se comentan las principales características de la misma y se invita al usuario a acceder a la aplicación.


Figura 3.18: Página principal de la plataforma

Tras acceder a la aplicación, lo primero que debe llevar a cabo el usuario es el registro [3.19] en la plataforma para poder utilizar a las funcionalidades de la misma.

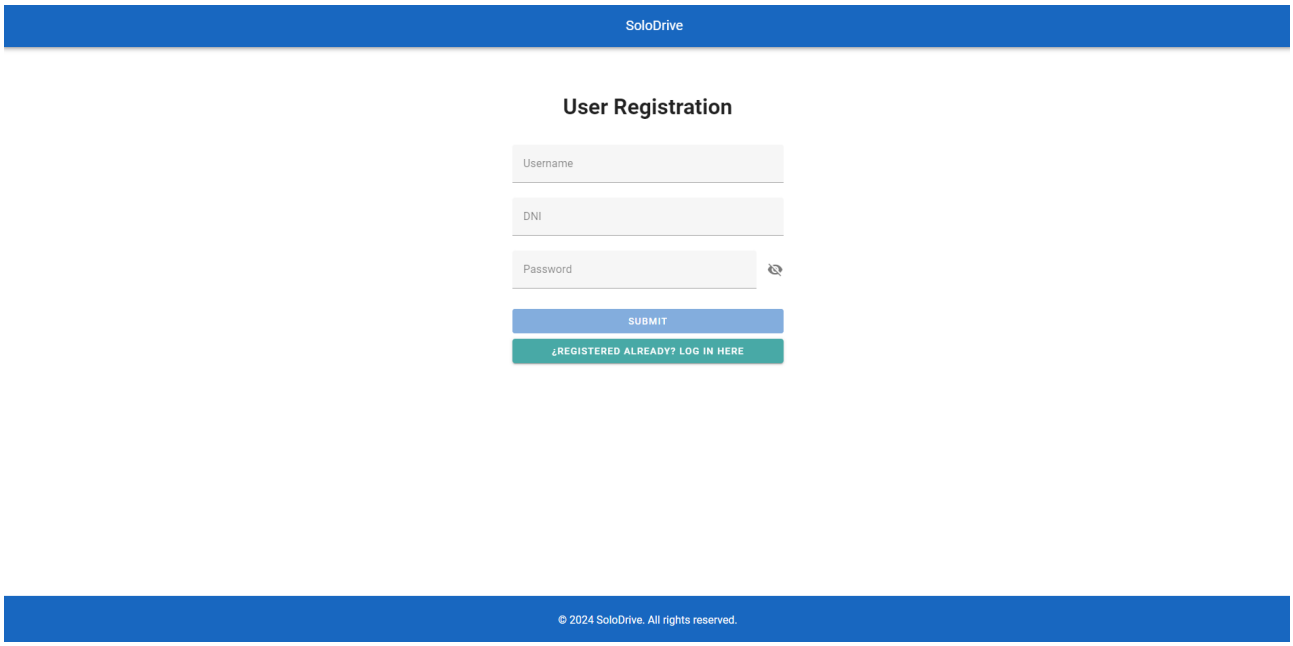



Figura 3.19: Formulario de registro de usuarios de la plataforma

En el caso de que el usuario ya se haya registrado previamente, deberá acceder [3.20] con sus credenciales a través del formulario de inicio de sesión.

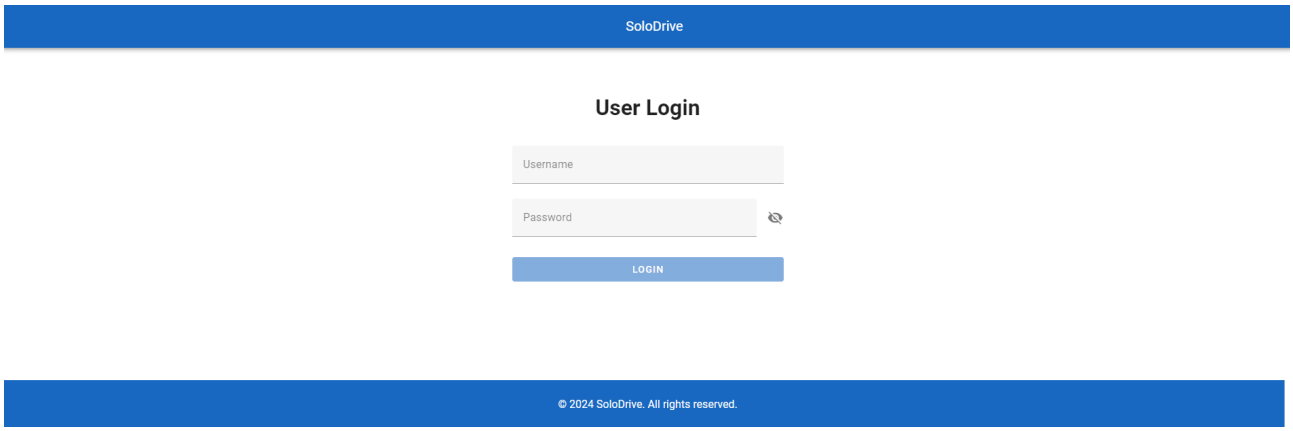



Figura 3.20: Formulario de inicio de sesión de la plataforma

En cualquiera de los casos, tanto si se crea una cuenta nueva como si se lleva a cabo el inicio de sesión con una cuenta existente, se llevará al usuario a la página donde se ubican los vehículos disponibles para el alquiler.

### 3.4.1. Vehículos disponibles

En esta sección se muestran todos aquellos vehículos que otros usuarios han decidido compartir a través de la plataforma y que están disponibles para alquilar. Tal y como se puede observar en [3.21], además de mostrarse los vehículos disponibles para alquilar, también se permite al usuario publicar vehículos si así lo desea.

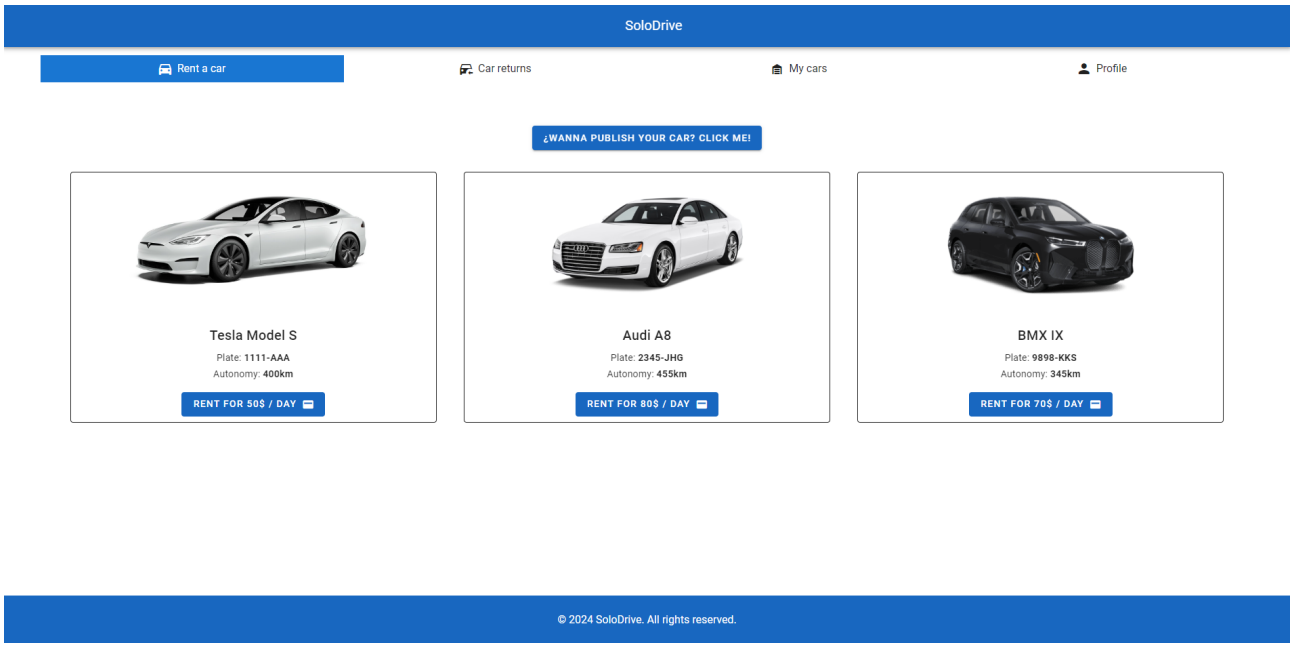



Figura 3.21: Vehículos disponibles para alquilar en la plataforma

### 3.4.2. Publicación de vehículos

Cuando un usuario quiere compartir su vehículo, debe publicarlo en la plataforma y para ello puede acceder al formulario de registro de vehículos [3.22] tanto a través de la página de vehículos disponibles que se acaba de comentar, como también a través del apartado de la barra de navegación que pone “My cars”, el cual hace referencia a la vista donde se muestran todos los vehículos que han sido compartidos por el usuario y además permite también al usuario publicar nuevos vehículos.

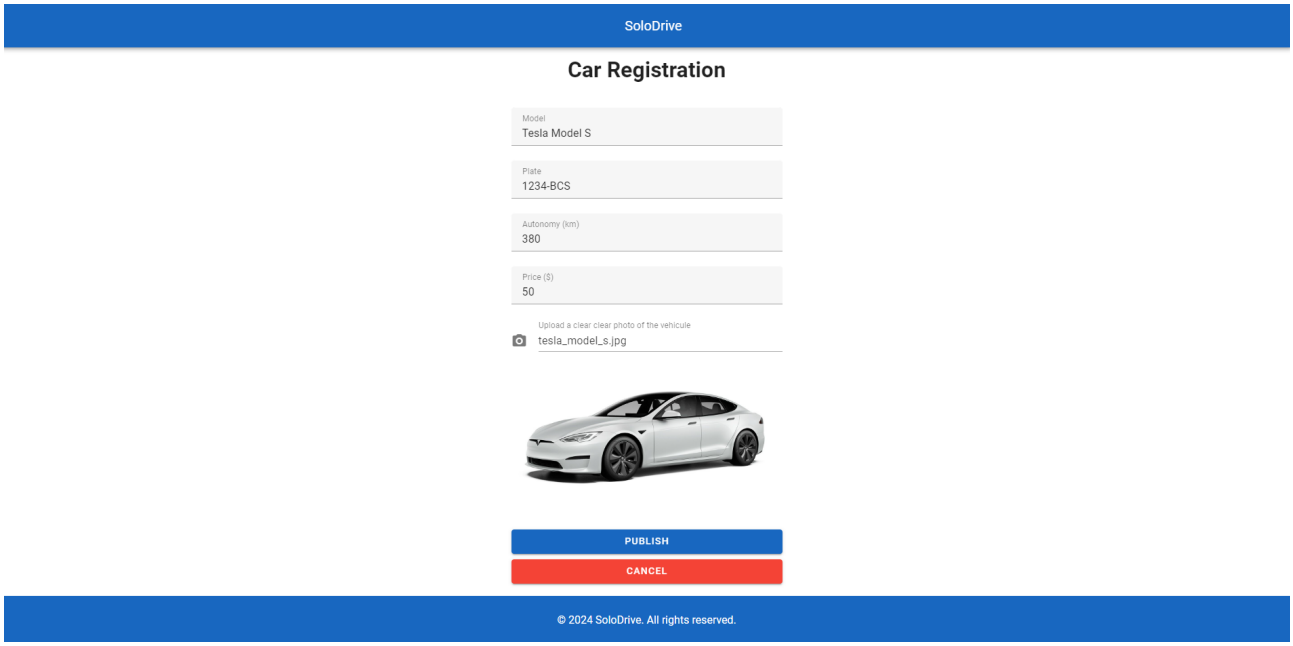



Figura 3.22: Formulario de registro de vehículos en la plataforma

### 3.4.3. Vehículos publicados

Como se ha comentado anteriormente, cuando un usuario comparte o publica sus vehículos, estos aparecen en la vista de “My cars”, la cual se muestra en la figura [3.23]. Adicionalmente, los usuarios tienen la posibilidad de quitar dichos vehículos de la plataforma en caso de que no quieran seguir compartiéndolos, siempre y cuando estos no estén en uso.

### 3.4.4. Devolución de vehículos

Cuando el usuario ha terminado de hacer uso del vehículo, deberá acceder a la vista de devolución de vehículos [3.24] para indicarlo. De esta manera el vehículo estará nuevamente disponible para ser alquilado por otros usuarios.

## 3.5. Detalles de Implementación Adicionales

En este apartado se comentan detalles de algunas implementaciones que se tuvieron que llevar a cabo para dar solución a algunos incovenientes que surgieron durante el desarrollo del proyecto.

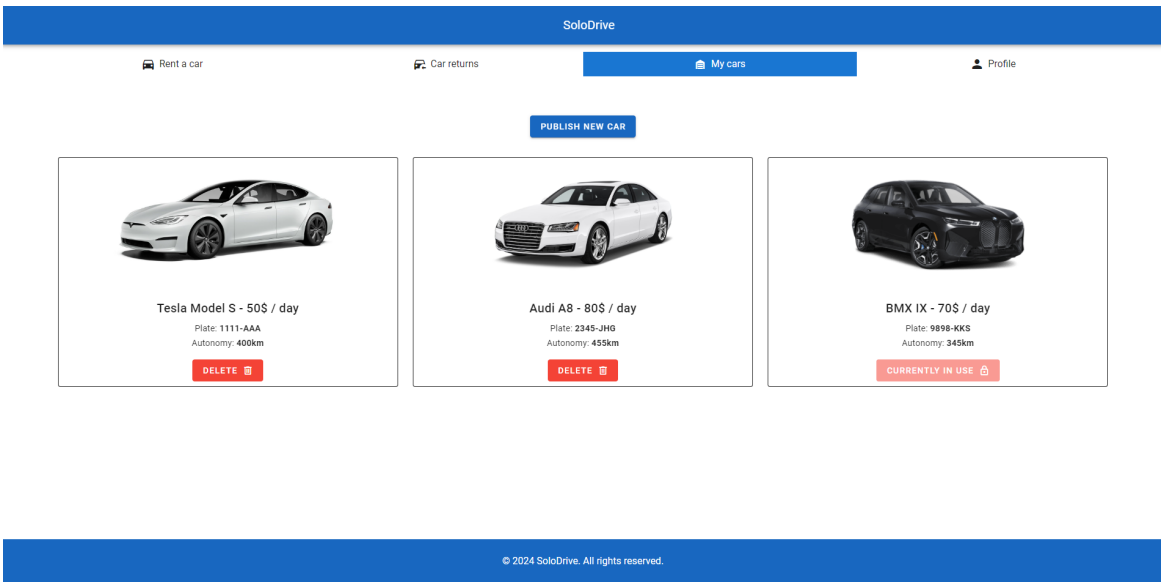



Figura 3.23: Vehículos publicados por el usuario en la plataforma

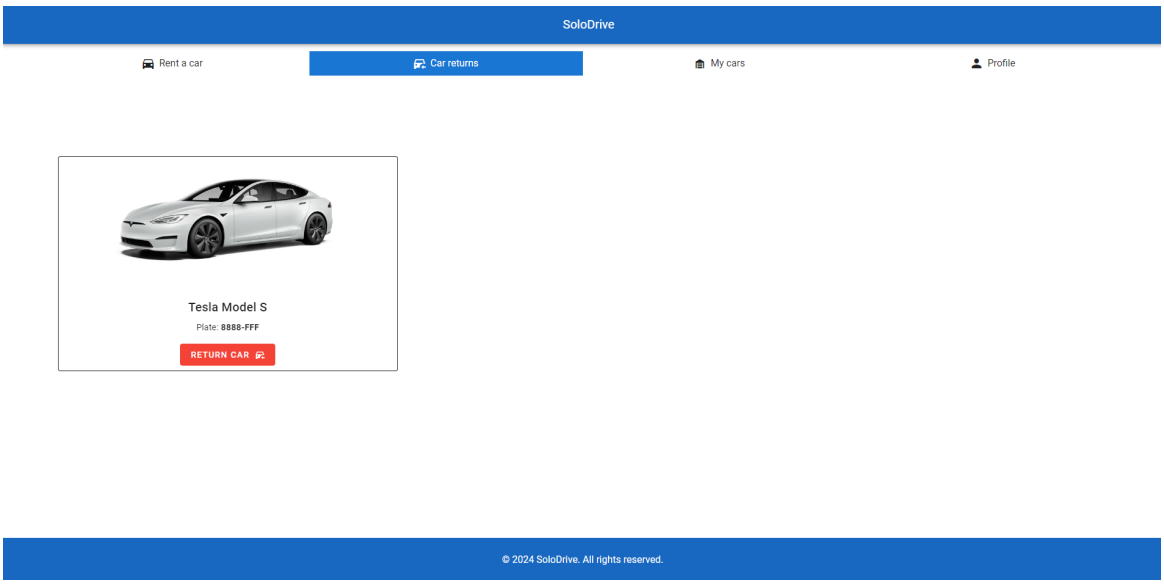



Figura 3.24: Vehículos publicados por el usuario en la plataforma

### 3.5.1. Conversión de monedas

Teniendo en cuenta que al publicar un vehículo, el precio establecido para el alquiler del mismo está en dólares, surge la necesidad de realizar una conversión de ese valor a su correspondiente en ether para poder enviar las cantidades correctas en las transacciones.

Para lograr este objetivo, se plantea el método `convertUSDToETH` ilustrado en la figura [3.25], en el que se hace uso de la API de CryptoCompare (24), la cual permite consultar el precio actual del par de divisas ETH/USD. Posteriormente, se realiza la conversión basada en la cantidad especificada de dólares que cuesta el alquiler del vehículo, asegurando así que las transacciones se efectúen con el monto adecuado en ether.

```
async function convertUSDtoETH(amountInUsd) {
  try {
    // Obtener el precio actual de ETH en USD
    const prices = await cc.price('ETH', ['USD']);
    const ethPriceInUsd = prices.USD;

    // Calcular la cantidad de ETH que puedes comprar con la cantidad especificada de USD
    const amountInEth = amountInUsd / ethPriceInUsd;

    return amountInEth;
  } catch (error) {
    console.error('Error fetching ETH price:', error);
    return null;
  }
}
```

Figura 3.25: Método para la conversión de dólares a ether

### 3.5.2. Almacenamiento de imágenes

En cuanto al almacenamiento de las imágenes, surgieron problemas debido a que no era viable utilizar la base de datos implementada para este propósito. Se consideró hacer uso de un servidor Amazon S3, pero debido a la falta de tiempo y para evitar complicar demasiado la implementación, se optó por usar el middleware Multer (25) de Node.js, que viene integrado en un módulo de NestJS. Este módulo, una vez configurado, permitía subir las imágenes sin problemas al servidor backend. Además, se añadió una configuración adicional al servidor para servir estas imágenes como contenido estático desde la ruta `/uploads`.

Por otro lado, se definieron una serie de reglas para optimizar el uso del almacenamiento dentro del servidor:

- Cuando se publica un vehículo con una imagen, la misma se renombra con la matrícula del vehículo. De esta manera, no se pueden publicar dos vehículos con la misma imagen porque Multer propagaría un error a través de NestJS.
- Cuando se elimina un vehículo, la imagen debe ser eliminada también para ahorrar almacenamiento y también para evitar que si se sube nuevamente un vehículo con esa matrícula, no se propague el error comentado con anterioridad desde Multer.

Adicionalmente, al crear un vehículo, como ya se conoce la ruta donde se servirá el contenido estático, se le asigna un campo al vehículo con la URL donde se encontrará su imagen correspondiente. Esto facilita la gestión y acceso a las imágenes, asegurando que cada vehículo tenga un enlace directo a su imagen en la ruta establecida.

# Capítulo 4

# Conclusiones y líneas futuras

En este capítulo se presentan las principales conclusiones extraídas del proceso de desarrollo, junto con propuestas de futuras mejoras que podrían enriquecer aún más la aplicación.

## 4.1. Conclusiones

La realización de este proyecto ha permitido profundizar en la tecnología blockchain aplicada al sector de la movilidad, en concreto, de los vehículos compartidos. Esta tecnología abre numerosas posibilidades debido a su capacidad para asegurar datos, automatizar acciones específicas a través de contratos inteligentes y operar sin necesidad de intermediarios.

El sistema seguro para compartir vehículos desarrollado en este trabajo representa lo que se considera una posible alternativa a la movilidad del futuro por varias razones:

- Al trabajar de manera exclusiva con vehículos autónomos, las rutas son planificadas de manera más eficiente, lo que ayuda a minimizar el consumo de energía y esto contribuye de manera significativa a la reducción de las emisiones de carbono y la contaminación ambiental.
- Introduce una propuesta innovadora al ya existente modelo de negocio “Peer-to-Peer car sharing” pero aplicado a los vehículos autónomos y que además está basado en la blockchain, lo que hace que sea un sistema único. Esta propuesta presenta numerosos beneficios:
    - **Economía colaborativa:** Los usuarios pueden alquilar sus vehículos autónomos cuando no están haciendo uso de ellos y sacar beneficios con la seguridad de que las condiciones del contrato de alquiler se van a cumplir.
    - **Interacción descentralizada:** Los usuarios que quieren hacer uso de los vehículos publicados pueden alquilarlos de manera rápida y directa, sin necesidad de esperar ningún tipo de aprobación por parte del propietario, ni de ningún intermediario, como si que suele suceder en el modelo convencional.
    - **Uso eficiente de recursos:** Se reduce la necesidad de poseer un vehículo propio, la cantidad de vehículos en circulación y se fomenta la utilización de vehículos que de otra manera estarían inactivos durante algún período de tiempo.

- Al gestionar los alquileres de los vehículos por servicio o día a través de un contrato inteligente, se elimina la posibilidad de que hayan deudas en los alquileres, porque los vehículos no se pueden utilizar hasta que no se efectua el pago y una vez terminan de usarlo y lo devuelven, este se bloquea, por lo que si quieren hacer uso del mismo de nuevo, deben volver a efectuar el pago. Esto también evita la necesidad de hacer uso de sistemas que pidan depósitos iniciales o fianzas para asegurar posibles impagos en el futuro.
- El uso de vehículos autónomos permite que en caso de accidente, el usuario no deba rendir ningún tipo de responsabilidad, ya que de deberse a un fallo del vehículo, esta recaería sobre la empresa fabricante del vehículo o sobre el propietario del mismo si es que no ha seguido un adecuado mantenimiento del mismo.

Todo esto sumado a las ventajas que da el uso de la tecnología blockchain y que ya se han comentado en apartados previos, hacen de esta plataforma una posible alternativa viable a los medios de transporte convencionales y a la movilidad del futuro.

## 4.2. Líneas futuras

A pesar de haber conseguido alcanzar con éxito los objetivos planteados, se plantean un serie de mejoras respecto del proyecto:

- **Aplicación Móvil:** Teniendo en cuenta que en la actualidad la mayoría de aplicaciones relacionadas con la movilidad son utilizadas desde teléfonos inteligentes, es casi un requisito obligatorio llevar a cabo una versión móvil de la aplicación.
- **Geolocalización:** En caso de querer hacer uso de un vehículo, es necesario poder localizar aquellos más cercanos, por lo que implementar un sistema de geolocalización solucionaría este problema.
- **Historial:** Sería interesante que los usuarios, tanto propietarios como clientes, pudieran consultar aquellos alquileres que han realizado en la aplicación, mostrando esa información a modo de historial.
- **Sistema de valoraciones:** Otro aspecto que podría dar más confianza a los usuarios sobre la aplicación, sería incluir un sistema de valoraciones donde los clientes puedan dar su opinión acerca de los vehículos que han alquilado, en cuanto a estado del vehículo y satisfacción con el trayecto realizado.
- **Alquiler por plazas de vehículo:** En el diseño actual de la aplicación, los vehículos son alquilados por completo, pero sería interesante tener en cuenta los trayectos que quieren realizar los clientes y en base a eso, permitir que compartan el mismo vehículo autónomo para llegar a sus destinos.

Aparte de la inclusión de todas estas mejoras y enfocando la plataforma hacia una movilidad más futurista, podría orientarse de manera que desde la propia aplicación móvil se tengan nociones de los vehículos autónomos cercanos, su precio, su estado de carga, las estaciones de carga más cercanas y las reseñas existentes sobre esos vehículos autónomos. Toda esta información permitiría a los usuarios hacer una criba para seleccionar aquel vehículo que consideren más apropiado para realizar su trayecto. Tras esto y la realización

del correspondiente pago, sería ideal que los vehículos autónomos pudieran también integrarse con la aplicación en cierta manera, para saber la ubicación del usuario y recogerlo en un punto específico para posteriormente llevarlos a su destino.

## 4.3. Agradecimientos

Esta investigación resulta de del Proyecto Estratégico SCITALA (C064/23), fruto del convenio de colaboración suscrito entre el Instituto Nacional de Ciberseguridad (INCIBE) y la Universidad de La Laguna. Esta iniciativa se realiza en el marco de los fondos del Plan de Recuperación, Transformación y Resiliencia, financiados por la Unión Europea (Next Generation).

# Capítulo 5

# Summary and Conclusions

This chapter presents the main conclusions drawn from the development process, along with proposals for future enhancements that could further enrich the application.

## 5.1. Conclusions

The implementation of this project has allowed to deepen the blockchain technology applied to the mobility sector, in particular, of shared vehicles. This technology opens up numerous possibilities due to its ability to secure data, automate specific actions through smart contracts, and operate without the need for intermediaries.

The secure car sharing system developed in this work represents what is considered a possible alternative to the mobility of the future for several reasons:

- By working exclusively with autonomous vehicles, routes are planned more efficiently, which helps minimize energy consumption and contributes significantly to the reduction of carbon emissions and environmental pollution.
- It introduces an innovative proposal to the already existing “Peer-to-Peer car sharing” business model but applied to autonomous vehicles and which is also based on blockchain, making it a unique system. This proposal presents numerous benefits:
    - **Collaborative economy:** Users can rent their autonomous vehicles when they are not using them and make a profit with the assurance that the terms of the rental contract will be met.
    - **Decentralized interaction:** Users who want to make use of the published vehicles can rent them quickly and directly, without having to wait for any type of approval from the owner or any intermediary, as is usually the case in the conventional model.
    - **Efficient use of resources:** It reduces the need to own a vehicle, reduces the number of vehicles on the road, and encourages the use of vehicles that would otherwise be idle for some period of time.
- By managing vehicle rentals by service or day through an intelligent contract, the possibility of debts in rentals is eliminated, because the vehicles cannot be used until payment is made and once they finish using it and return it, it is blocked, so that if they want to use it again, they must make the payment again. This also avoids

the need to make use of systems that ask for initial deposits or deposits to ensure possible non-payments in the future.

- The use of autonomous vehicles means that in the event of an accident, the user does not have to bear any type of responsibility, since if it is due to a vehicle failure, the responsibility would fall on the vehicle manufacturer or on the owner of the vehicle if it has not been properly maintained.

All this added to the advantages of blockchain technology, which have already been discussed in previous sections, make this platform a possible viable alternative to conventional means of transportation and the mobility of the future.

## 5.2. Future lines

Despite having successfully achieved the objectives set, a series of improvements to the project are proposed:

- **Mobile Application:** Taking into account that nowadays most of the applications related to mobility are used from smartphones, it is almost a mandatory requirement to carry out a mobile version of the application.
- **Geolocation:** In case you want to make use of a vehicle, it is necessary to be able to locate those closest to you, so implementing a geolocation system would solve this problem.
- **History:** It would be interesting that users, both owners and customers, could consult those rentals they have made in the application, showing that information as a history.
- **Rating system:** Another aspect that could give users more confidence in the application would be to include a rating system where customers can give their opinion about the vehicles they have rented, in terms of vehicle condition and satisfaction with the journey made.
- **Rental by vehicle seats:** In the current design of the application, vehicles are fully rented, but it would be interesting to take into account the journeys that customers want to make and based on that, allow them to share the same autonomous vehicle to reach their destinations.

Apart from the inclusion of all these improvements and focusing the platform towards a more futuristic mobility, it could be oriented in such a way that the mobile application itself would provide information on nearby autonomous vehicles, their price, their state of charge, the nearest charging stations and the existing reviews on these autonomous vehicles. All this information would allow users to make a sieve to select the vehicle they consider most appropriate for their journey. After this and the corresponding payment, it would be ideal if the autonomous vehicles could also be integrated with the application in some way, to know the user's location and pick them up at a specific point and then take them to their destination.

## 5.3. Acknowledgements

This research is the result of the SCITALA Strategic Project (C064/23), fruit of the collaboration agreement signed between the National Cybersecurity Institute (INCIBE) and the University of La Laguna. This initiative is carried out within the framework of the funds of the Recovery, Transformation and Resilience Plan, financed by the European Union (Next Generation).

# Capítulo 6

# Presupuesto

En este capítulo se presentan de manera estimada los costes asociados al proyecto, distinguiendo por un lado los costes relativos a recursos humanos y por otro, los costes tecnologícos.

## 6.1. Costes de Recursos Humanos

Se presentan en la tabla [6.1] los costes asociados a los recursos humanos teniendo en cuenta las horas empleadas para cada tipo distinto de tarea.

| Tipos | Duración | Coste |
|---|---|---|
| Estudio de blockchain | Total de 24 horas | 40€/h |
| Estudio del estado del arte | Total de 12 horas | 40€/h |
| Estudio de tecnologías | Total de 15 horas | 40€/h |
| Trabajo de desarrollo | Total de 115 horas | 40€/h |

Tabla 6.1: Costes de Recursos Humanos del Proyecto

El coste total asociado a los recursos humanos empleados para el desarrollo del proyecto es de 6800€.

## 6.2. Costes Tecnológicos

Teniendo en cuenta un lanzamiento a producción de la aplicación durante un período de 3 meses, se necesitarían una serie de recursos para garantizar el correcto funcionamiento de la aplicación, cuyos costes se desglosan en la tabla [6.2].

| Tipos | Duración | Coste |
|---|---|---|
| Servidor para alojar frontend y backend | Total de 3 meses | 100€/mes |
| Nodo de blockchain | Total de 3 meses | 50€/mes |

Tabla 6.2: Costes Tecnológicos del Proyecto

El coste total asociado a los recursos tecnológicos empleados para el desarrollo del proyecto es de 450€.

## 6.3. Costes Totales

El coste total del proyecto es la suma de los costes de recursos humanos y los costes tecnológicos, detallados en las tablas anteriores.

| **Tipo** | **Coste** |
|---|---:|
| Recursos Humanos | 6800€ |
| Recursos Tecnológicos | 450€ |
| **Total** | **7250€** |

Tabla 6.3: Costes Totales del Proyecto

El coste total del proyecto es de 7250€.

# Bibliografía


[1] Haber, Stuart and Stornetta, W. Scott, “How to time-stamp a digital document,” *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991.

[2] Nakamoto, Satoshi, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] Buterin, Vitalik, “Ethereum: A next-generation smart contract and decentralized application platform,” 2013. [Online]. Available: https://ethereum.org/en/whitepaper/

[4] NIST, “Cryptographic hash function - glossary.” [Online]. Available: https://csrc.nist.gov/glossary/term/cryptographic_hash_function

[5] Ethereum, “The merge.” [Online]. Available: https://ethereum.org/en/roadmap/merge/

[6] Szabo, Nick, “Smart contracts: Building blocks for digital markets,” 1996. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[7] DAV, “Decentralized autonomous vehicles.” [Online]. Available: https://dav.network/

[8] Helbiz, “One tap away.” [Online]. Available: https://helbiz.com/es

[9] Torres, Néstor, “SoloDrive.” [Online]. Available: https://github.com/dtote/tfm-solodrive

[10] Vue.js, “Vuejs.org.” [Online]. Available: https://vuejs.org/

[11] Vuetify Team, “Vuetify.” [Online]. Available: https://vuetifyjs.com/

[12] MongoDB, Inc., “Mongodb - the database for modern applications.” [Online]. Available: https://www.mongodb.com/

[13] ECMA International, “Javascript - ecmascript language specification.” [Online]. Available: https://262.ecma-international.org/

[14] NestJS, “Nestjs - a progressive node.js framework.” [Online]. Available: https://nestjs.com/

[15] Node.js, “Node.js - javascript runtime built on chrome’s v8 javascript engine.” [Online]. Available: https://nodejs.org/

[16] TypeScript, “Typescript - javascript with syntax for types.” [Online]. Available: https://www.typescriptlang.org/

[17] NestJS, “@nestjs/mongoose.” [Online]. Available: https://www.npmjs.com/package/@nestjs/mongoose

[18] Solidity, “Solidity - ethereum smart contract programming language.” [Online]. Available: https://soliditylang.org/

[19] Truffle Suite, “Truffle - smart contract development framework.” [Online]. Available: https://trufflesuite.com/truffle

[20] ——, “Truffle suite - development environment, testing framework and asset pipeline for ethereum.” [Online]. Available: https://trufflesuite.com/

[21] ——, “Ganache - personal blockchain for ethereum development.” [Online]. Available: https://trufflesuite.com/ganache

[22] Web3.js, “Web3.js - ethereum javascript api.” [Online]. Available: https://web3js.readthedocs.io/

[23] MetaMask, “Metamask - a crypto wallet & gateway to blockchain apps.” [Online]. Available: https://metamask.io/

[24] CryptoCompare, “Cryptocurrency api, historical & real-time market data.” [Online]. Available: https://min-api.cryptocompare.com/

[25] Npm, “Multer.” [Online]. Available: https://www.npmjs.com/package/multer