Trabajo de Fin de Grado

Aplicación web descentralizada para el alquiler de vehículos en aeropuertos

Descentralized web application for car rental at airports

Néstor García Moreno

La Laguna, 1 de junio de 2018

Dña. **Pino Caballero Gil** , con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **José Ángel Concepción Sánchez**, con N.I.F. 42.234.897-C Personal Investigador adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

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

Que la presente memoria titulada:

"*Aplicación web descentralizada para el alquiler de vehículos en aeropuertos*"

ha sido realizada bajo su dirección por D. **Néstor García Moreno**, con N.I.F. 79.085.553-F.

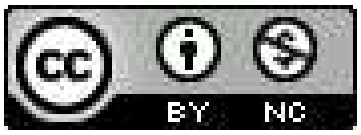
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 1 de junio de 2018

Agradecimientos

Agradecerle a mi tutora, Pino Caballero Gil, y a mi cotutor José Ángel Concepción Sánchez por sus apoyos y haberme guiado en todo momento. Sus consejos y ayuda han sido vitales para la consecución de este proyecto.

A mi familia por apoyarme durante toda la carrera de Ingeniería Informática.

Licencia



Resumen

El objetivo de este trabajo ha sido una investigación exhaustiva sobre las plataformas Bitcoin y Ethereum, enfatizando en el campo de la criptografía y de toda la tecnología que hay detrás de estas plataformas como Blockchain, nodos, transacciones, consenso, contratos inteligentes, algoritmo de prueba de trabajo, etc.

Se ha aplicado el conocimiento de la investigación en el desarrollo de una aplicación web descentralizada para el alquiler de vehículos siguiendo las directrices de Ethereum. El doble fin de esta aplicación es, por tanto, la demostración práctica de estos conocimientos, cubriendo todo el proceso de alquiler de vehículos de las aplicaciones web tradicionales, dotando el desarrollo de autonomía y facilidad para los usuarios finales.

Por último, para el desarrollo de la aplicación se ha seguido el Modelo Vista Controlador, de forma que toda la lógica de negocio recae en los contratos inteligentes implementados en la red Ethereum donde estos contratos controlan todo el sistema de alquiler de vehículos de los clientes.

Palabras clave: Blockchain, Ethereum, Contratos Inteligentes, Aplicaciones Descentralizadas, Alquiler de Vehículos, JavaScript

Abstract

The objective of this work has been an exhaustive research on the Bitcoin and Ethereum plataforms, emphasizing in the field of cryptograph and all the technology behind these platforms like Blockchain, nodes, transactions, consensus, smart contracts, proof of work algorithm, etc.

We have applied the knowledge of the research in the development of a decentralized web application for the rent of vehicles following the Ethereum guidelines. The double purpose of this application is, therefore, the demonstration practice of this knowledge for the coverage throughout the vehicle rental process of traditional web applications, endowing it with autonomy and ease for users.

Finally, for the development of the application the View Controller Model has been followed, so that all the business logic falls on the the smart contracts implemented in the Ethereum network where these contracts control the entire system to rent vehicles of customers.

Keywords: *Blockchain, Ethereum, Smart Contracts, Decentralized Applications, Rent a Car, JavaScript*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

El uso de la tecnología Blockchain en este proyecto supone una oportunidad única para aprender y comprender toda la tecnología en materia de seguridad que hay detrás de la cadena de bloques y sus aplicaciones derivadas, como las criptomonedas o los contratos inteligentes. Me apasiona el campo de la seguridad informática y la criptografía, que han facilitado la creación de la Blockchain gracias a la criptografía asimétrica y a las funciones hash, ambas detalladas más adelante.

En los últimos años, ha tenido lugar un auge de esta tecnología, viéndose incrementada su popularidad y uso. Dicho auge ha sido motivado mayormente por las criptodivisas, pero también por los diferentes usos de los contratos inteligentes y la cadena de bloques. Su estandarización podría provocar una revolución de la red 3.0, estableciendo una descentralización real de la información y de las aplicaciones.

El presente Trabajo de Fin de Grado presenta uno de los múltiples usos que tienen los contratos inteligentes, como entidades seguras, independientes y descentralizadas que sirven para sellar un acuerdo entre dos o más partes, en este caso, entre una empresa de alquiler de coches y sus clientes. La implementación real de este contrato podría facilitar todo el proceso de arrendamiento de vehículos, por lo que la principal motivación ha sido crear una aplicación descentralizada como alternativa a las clásicas compañías de alquiler, dotándola de todas las características propias de su sector y añadiéndole un valor distinguible entre sus competidores.

1.2. Objetivos

Como ya se comentó en el apartado anterior, el principal objetivo de este proyecto es hacer uso de los contratos inteligentes, aprovechando la tecnología que hay detrás del Blockchain público de Ethereum para construir una apli-

cación descentralizada y segura, utilizando el token de Ethereum (ether), y el algoritmo de prueba de trabajo para las transacciones (incluyendo todos los procesos del alquiler de coches: el alquiler, la devolución, añadir fondos, etc.). Esta aplicación es capaz de responder a todas las necesidades clásicas de un sistema de arrendamiento e ir más allá para crear una aplicación inteligente que automatice cualquier proceso entre el cliente y la compañía, aumentando su accesibilidad y usabilidad.

Un segundo objetivo es entender la tecnología criptográfica detrás de Ethereum y Bitcoin y comprender los múltiples usos que podría tener, así como sus posibilidades a largo plazo.

1.3. Fases del desarrollo

Las fases del desarrollo del presente trabajo se pueden agrupar en cinco etapas diferentes.

Una primera fase o etapa se dedica al estudio para comprender y entender toda la tecnología Bitcoin y Ethereum (nodos, transacciones, prueba de participación, prueba de esfuerzo o de trabajo, doble gasto, latencia, cadena de bloques, bloque, consenso, etc) y las diferentes características entre sendas plataformas.

Una segunda fase fue el aprendizaje, orientada a la documentación sobre los contratos inteligentes de una manera teórica (fundamentos, aplicaciones, seguridad, normalización, etc.) y de manera práctica con el uso de los dos lenguajes de programación utilizados, Solidity y Serpent.

Se continúo con un análisis de todas las herramientas para desarrollar aplicaciones descentralizas, comparando sus ventajas y desventajas. En esta fase también se realizó una primera aproximación del diseño de la aplicación final.

Una vez elegidas las herramientas necesarias, en la cuarta fase se construyó la plataforma, se mejoró el diseño anterior, y se definieron las bases de datos, y todas las funcionalidades de la misma.

En la etapa final se automatizaron los procesos de cobro de los contratos con el framework Ethereum Alarm Clock.

1.4. Estructura de la memoria

El presente documento está estructurado en ocho capítulos, donde el primero es de introducción al trabajo, detallando los objetivos y motivaciones, las etapas del desarrollo del proyecto y la estructura de este documento.

Seguido del primer capítulo, se definen todos los fundamentos teóricos de seguridad y criptografía que hay detrás de la tecnología Bitcoin y Ethereum, debido a su importancia y peso en este trabajo, como ya se comentó.

En tercer lugar, se realiza un análisis de las herramientas más preclaras en la

comunidad para el desarrollo de este tipo de aplicaciones, así como un sondeo de las aplicaciones más cercanas a nuestra aplicación.

Tras ello, se encuentra la piedra angular de la memoria, el cuarto capítulo, pues se detalla profundamente la aplicación, sus contratos, la implementación de la tecnología, los diferentes ficheros y directorios, funciones y métodos. También se exponen componentes de código reseñables de la aplicación y su explicación algorítmica.

Por otro lado, toda aplicación requiere de un testing, pruebas, en este caso, unitarias. Para comprobar el correcto funcionamiento de la misma, debe abarcar el máximo conjunto de código posible, así como los problemas encontrados y soluciones propuestas, y las ventajas y desventajas de estas soluciones. Todas estas inquietudes quedan recogidas en el quinto capítulo.

Asimismo, todo proceso de software tiene asociado unos costes, humanos y materiales y que están descritos y tabulados en el séptimo capítulo.

Por último, los capítulos finales son las conclusiones y líneas futuras de trabajo tanto en español como en inglés, donde se recoge un extracto a modo de resumen del trabajo y posibles implementaciones del trabajo futuras para continuar con la línea de desarrollo y de investigación.

Capítulo 2

Análisis de la tecnología blockchain

Esta sección abordará el estudio de la tecnología que hay detrás de Blockchain, y las redes Bitcoin y Ethereum.

2.1. Bitcoin

En 2008 Satoshi Nakamoto describió en el white paper "Bitcoin: Un Sistema de Efectivo Electrónico Usuario-a-Usuario", los fundamentos del Bitcoin, transacciones, algoritmo de prueba de trabajo y Blockchain.

Se trata de una red, un protocolo, basada en una red entre iguales (peer-to-peer), donde todos los nodos son iguales entre sí, prescindiendo de la figura de un nodo central o servidor, para evitar la centralización de la red.

Satoshi Nakamoto propone en la red Bitcoin la moneda con el mismo nombre para crear una alternativa a los sistemas financieros tradicionales. Estos sistemas tradicionales, requieren de un tercero confiable (banco o entidad) para el procesamiento de pagos electrónicos. Esto genera una vulnerabilidad, ya que si se ataca al nodo central todo el sistema cae. También al ser una entidad confiable, los usuarios deben autenticarse de alguna forma y son dependientes de ella y cualquier transacción o movimiento debe pasar por la entidad previamente autenticado.

Para la red descentralizada de Nakamoto, se usan las funciones hash criptográficas y la criptografía de clave pública.

2.2. Criptografía

Las redes Bitcoin y derivadas como Ethereum, usan para el desarrollo de sus plataformas los siguientes campos de la criptografía.

2.2.1. Funciones hash

Las funciones hash son utilizadas en la tecnología Blockchain debido a que proporcionan un valor aleatorio de longitud fija a partir de una entrada arbitraria. Este proceso no es computacionalmente reversible, por lo que dado un valor hash es prácticamente imposible obtener el dato original. Las funciones hash ampliamente utilizadas actualmente, SHA-2 y SHA-3 tienen una probabilidad de colisión extremadamente baja, es decir, que dos entradas den la misma salida es prácticamente imposible. Bitcoin usa SHA256 que retorna siempre una cadena de 64 caracteres hexadecimales, es decir, 256 bits de información.

Estas funciones hash se utilizan para verificar la integridad de datos, para comprobar si han sido modificados o no, ya que el valor hash asociado correspondiente cambia si la entrada cambia. El mismo dato siempre producirá el mismo hash.

2.2.2. Criptografía de clave pública

Asegura las transacciones ejecutadas en muchas criptodivisas, incluidas Bitcoin y Ether. Un usuario tiene una clave pública, disponible para cualquier persona y una clave privada que debe mantenerse en secreto. Esta clave se usa para firmar transacciones y con la clave pública se verifica el remitente de estas transacciones.

En la criptografía asimétrica la clave privada se usa para descifrar y la clave pública para cifrar los mensajes (o transacciones). Es una función unidireccional: transformación de fácil aplicación, pero de difícil inversión (ver Figura 2.1).

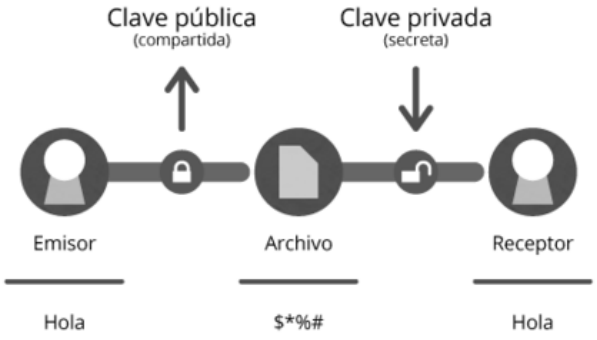


Figura 2.1: Clave pública

2.3. Usuarios

Para que la red sea anónima no puede haber ninguna entidad que lleve un control o un registro de los usuarios, para ello se usan direcciones hexadecimales donde cada usuario crea la suya propia usando la criptografía asimétrica de

manera autónoma e independiente. La clave pública se usa para recibir los pagos y la clave privada para obtener los fondos.

Tal como se ha descrito, se usan direcciones hexadecimales en vez de claves públicas para diferenciar a los usuarios por comodidad. La dirección hexadecimal es un resumen de la clave pública haciéndola menos extensa.

2.4. Transacciones

Una transacción es cualquier envío de moneda digital a un destinatario. Para ello, el dueño de la moneda debe firmar con un hash de la última transacción conocida y la clave pública del receptor del dinero como se muestra en la figura 2.2. Las transacciones están compuestas por una o más entradas, una o más salidas y muchos productos transaccionales no gastados, Usnpent Transaction Outputs (UTXOs). Se usan para comprobar si la transacción origen es válida y para ello, los nodos guardan en caché las transacciones que no han sido gastadas de manera listada para comprobar a posteriori si la transacción que se quiere usar está en esta tabla.

En cada transacción el emisor puede dejar una comisión opcional para agilizar el proceso de minado del bloque que a continuación comentaremos.

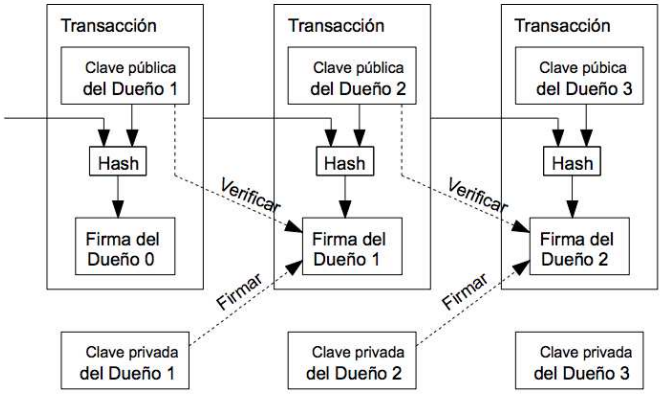


Figura 2.2: Firmando una transacción

2.5. Blockchain

Blockchain es el gran libro contable de la red, es una base de datos distribuida, pública, conformada por bloques. Cada bloque individual contiene su valor hash, una o más transacciones, donde las transacciones pueden ser envío de dinero, un contrato inteligente o simplemente texto, y el nonce asociado a dicho bloque. El nonce es un campo de 4 bytes cuyo valor se establece para que el hash asociado contenga una secuencia de ceros a la izquierda, inicialmente Bitcoin partía de 9

ceros en el Hash, a día de hoy va por 16, y va creciendo de manera exponencial para aumentar la complejidad.

Si cambia el nonce o el campo de la información, la función hash cambiará (ver Figura 2.3).

Figura 2.3: Ejemplo de un bloque

Para introducir el bloque en la cadena de bloques ha de ser minado. Minar un bloque significa encontrar encontrar una función hash menor a un número objetivo usando el algoritmo de prueba de trabajo que se explicará más adelante.

Una vez el bloque, que contiene varias transacciones, ha sido minado es introducido en la cadena de bloques, las transacciones que no están dentro de la Blockchain no se consideran válidas. El bloque introducido en la cadena contiene también una función hash del bloque anterior, y este del anterior, formando la cadena de bloques. (ver Figura 2.4).

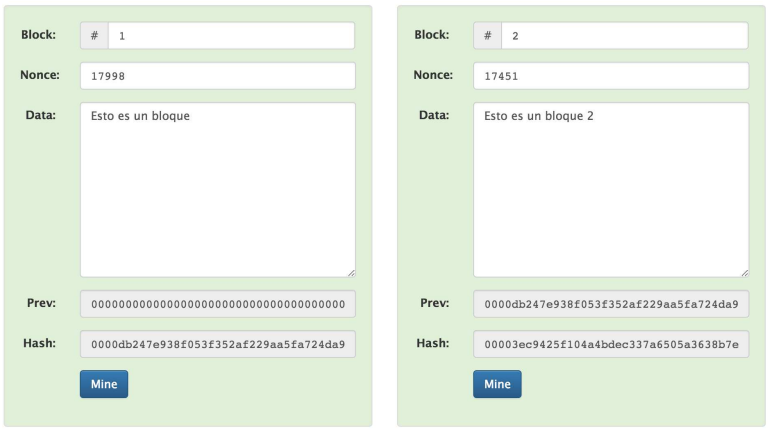


Figura 2.4: Cadena de bloques

Cada bloque también contiene un campo 'Cadena de Merkle' que proporciona de manera eficiente todas las funciones hash de todas las transacciones incluidas en el bloque.

2.6. Prueba de trabajo

Como se comentó en el apartado anterior, para que un bloque sea introducido en la cadena de bloques ha de ser minado. Para ello se usa el algoritmo de prueba de trabajo o de esfuerzo. Este algoritmo garantiza, en cierta medida que la Blockchain no pueda ser reescrita. Es un algoritmo muy costoso en tiempos computacionales, ya que requiere encontrar un valor nonce adecuado a la función hash del bloque y este valor debe estar por debajo de un cierto valor (número objetivo o umbral). Se necesitan muchos intentos para conseguir el nonce correcto y usar la función hash SHA-256 para calcular el hash del bloque minado.

Este número objetivo es ajustado dinámicamente para que un nonce sea encontrado aproximadamente en unos 10 minutos. Satoshi estableció que este algoritmo crecía exponencialmente haciendo más costoso minar bloques. Cada bloque que se mina es más costoso de minar que el anterior. También calculó la probabilidad de que un atacante genera una cadena alterna más rápida que la cadena honesta y para que la red sea atacada debe haber más del 51 % de nodos controlados por atacantes. Sin embargo, si esto ocurre y se modifica la cadena de bloque malintencionadamente hace que el precio del bitcoin caiga, por lo que se incentiva a permanecer veraces en el minado de bloques. Las probabilidades de que esto ocurra pueden encontrarse en libro blanco de Satoshi y son realmente bajas [19].

En cada bloque minado se genera un coinbase para el nodo que ha minado dicho bloque. Así es como se generan nuevos tokens, siendo el algoritmo de Bitcoin inflacionista y el de Ethereum deflacionista, en Bitcoin el número de bitcoins está limitado y establecido desde el primer día en 21 mil millones de tokens y el de Ethereum no tiene valor límite.

2.7. Consenso

Puede darse el caso de que en la cadena de bloques, hayan varios nodos a la vez minando bloques de manera simultánea. La cadena seguirá creciendo y los nodos irán replicando esta información, los bloques nuevos han sido minados y ya están en la cadena de bloques, pero al haber sido introducidos a la vez puede que haya información contradictoria en los bloques, cuando se dé la colisión y haya un nodo con los bloques nuevos y la información sea discordante ganará aquel bloque que esté dentro de una cadena de bloques mayor.

Se propagará esta nueva información descartando el otro bloque. Se replicará por el resto de la cadena, y los nodos adyacentes también descartarán el bloque que previamente ha sido minado, de modo que estas transacciones deberán volver a ser minadas.

2.8. Cadena de Merkle

Es un árbol binario donde los nodos interiores, es decir, los nodos que no son terminales contienen la función hash de las funciones hash de sus hijos y así respectivamente hasta llegar hasta el nodo raíz del árbol llamado Merkle root. Esta estructura en forma de árbol es una forma eficiente de almacenar datos que pueden ser comprobados (ver Figura 2.5).

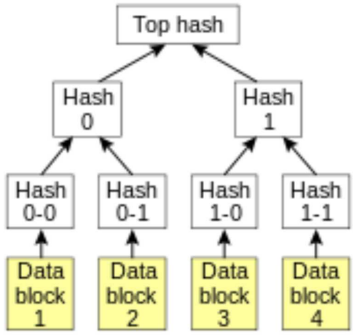


Figura 2.5: Cadena de Merkle

2.9. Inconvenientes

2.9.1. Doble gasto

Se conoce como doble gasto a intentar enviar dos fondos monetarios a distintos receptores en sendas transacciones. La única transacción válida será la primera en ser minada, esta transacción al validarse descarta la otra y se propaga por la red. Si aún así una persona quisiera hacer el doble gasto con una transacción ya validada en la cadena de bloques debería minar otra transacción para alterar la ya introducida, e introducir en la cadena de bloques, y propagarla por la red. Cuando las dos cadenas con cada transacción respectivamente lleguen a un nodo común y se produzca la colisión, para que se tome como válida debería tener el atacante el 51 % de la red, tal y como se describió en el apartado 2.6.

2.9.2. Latencia

El principal problema de las redes que utilizan Blockchain es la latencia, tiempo que tarda desde que se produce una transacción hasta que esta es válida, es decir, es introducida en la cadena de bloques. Como se comentó, tarda de media unos 10 minutos en minarse un bloque, pero aún así tiene este bloque que propagarse por los nodos adyacentes y expandirse por toda la cadena para que en caso de colisión, el algoritmo de consenso la tome como válida.

Esto provoca que las redes descentralizadas sean incompatibles con procesos que requieran de inmediatez como pueden ser la bolsa o las subastas.

2.10. Ethereum

Ethereum es otra red, plataforma y protocolo que comparte muchos de los conceptos con Bitcoin, como la cadena de bloque, las transacciones, el consenso, un algoritmo de prueba de trabajo. También contiene cosas propias como el algoritmo de participación, que es el algoritmo que se usará para minar bitcoins a partir de 2020, donde en vez de minar usando toda la fuerza bruta computacional del nodo sería apostando o confiando parte de tus ethers en la transacción junto a otros nodos, de modo que se validarán las transacciones con mayor número de nodos apostando por ellas.

Sin embargo, la novedad más importante que añade Ethereum es el concepto de contrato inteligente, para ser ejecutado en los nodos de la red. Estos contratos inteligentes pueden ser desarrollados por cualquier persona que tenga un mínimo de conocimiento de programación.

Un contrato inteligente es un programa que sella un acuerdo entre dos o más entidades sin necesidad de una tercero de confianza. Se le añaden entradas al contrato y la condición de contrato para que se cumpla. En la ejecución, una una vez que se cumpla la condición, se entregan las salidas a la entidad indicada. Esta tecnología permite la creación de aplicaciones descentralizadas, puesto que la lógica de negocio recae en la Blockchain.

La moneda que se usa para las transacciones es el ether, pero también se incluyen otras unidades partición del ether para la comodidad de las transacciones.

Unidad	**Valor Wei**
wei	1 wei
Kwei (babbage)	1e3 wei
Mwei (lovelace)	1e6 wei
Gwei (shannon)	1e9 wei
microther (szabo)	1e12 wei
milliether	1e15 wei
ether	1e18 wei

Tabla 2.1: Tabla de unidades Ether

El gas es la unidad de cálculo con un precio asociado en ether, que se usa para los cálculos de los contratos inteligentes con el fin de evitar fines maliciosos en los contratos como estructuras complejas innecesarias, bucles infinitos, etc. A mayor complejidad del contrato inteligente mayor cantidad de gas habrá que gastar para ejecutar el contrato y por tanto mayor cantidad de ether.

Capítulo 3

Antecedentes y estado actual

Se abordan los antecedentes y el estado actual de toda la tecnología Blockchain, Bitcoin y Ethereum, así como un análisis de aplicaciones cercanas a la desarrollada y la tecnología requerida para hacer estas aplicaciones.

3.1. Estado del arte

El 3 de Enero de 2009 se generó el primer bloque de la Blockchain de Bitcoin, el bloque Génesis, marcando el inicio de esta red. Seis días más tarde salía al mercado el bitcoin, un nuevo sistema de dinero digital con el mismo nombre que la red, y, donde el doble gasto no era posible. Nueve años más tarde, ya existen en el mercado 1610 criptodivisas. Las monedas criptográficas no son la única aplicación de la Blockchain, pero es la aplicación más popular y el aumento exponencial del mercado de criptodivisas nos da una idea del crecimiento del Blockchain y su tecnología.

Por su parte, Ethereum sale al mercado el 30 de julio de 2015 y va más allá de la red Bitcoin. Es un protocolo, una plataforma, una criptomoneda (ether) y un lenguaje de programación cuyo objetivo principal es la creación de contratos inteligentes. Desde entonces se han minado más de 1 millón de contratos inteligentes (1 millón de direcciones hexadecimales únicas pertenecientes a contratos). Sus fines son muy diversos, desde el alquiler de coches como en nuestro caso, apuestas y/o loterías, acuerdos de compra/venta, herencias o cualquier otro proceso de notaría, etc.

Este tipo de contratos inteligentes podría llegar a reemplazan a los contratos tradicionales. Para ello, deberán estandarizarse con ayuda de los organismos públicos y los gobiernos. A diferencia de estos, las grandes empresas mundiales ya han empezado a apostar por estas tecnologías, como puede evidenciarse en la creación del Ripple, una criptomoneda bancaria que nace de la unión de los grandes bancos más importantes del mundo.

3.2. Aplicaciones

3.2.1. HireGo

Plataforma descentralizada para compartir coches que saldrá al mercado en el último cuatrimestre del 2018. En su cronograma destaca el lanzamiento de su propio token para usar su servicio y la implementación de Internet de las Cosas y los coches inteligentes para automatizar las transacciones junto a su token. Es la primera aplicación descentralizada en la red de pruebas de Ethereum (Test Net) de compartición de coches (ver Figura 3.1).

Figura 3.1: Plataforma HireGo [6]

3.2.2. Darenta

Aplicación idéntica a HireGo pensada para el alquiler de coches de particulares usando también su propio token. Según su whitepaper el token debería haber salido hace meses, por lo que es una aplicación aún prototipada y teórica. Como HireGo, no han salido a la Blockchain pública de Ethereum. Darenta quiere ser el AirBnB de coches, creando comunidades de usuarios para ganar dinero alquilando su propio coche a terceros mediante el uso de contratos inteligentes (ver Figura 3.2).

Figura 3.2: Plataforma Darenta [2]

3.2.3. Helbiz

Helbiz es una aplicación móvil y web para alquilar tu propio vehículo privado y ganar dinero usando su propia criptomoneda (HelbizCoin). Toda las transacciones se gestionan a través de su aplicación móvil. A diferencia de las otras aplicaciones su moneda ya ha salido al mercado como token y la quieren estandarizar. Helbiz ya está operativa y funcional en las redes públicas de Ethereum (ver Figura 3.3).

Figura 3.3: Plataforma Helbiz [5]

3.3. Tecnologías

A continuación se detallan algunas de las tecnologías necesarias para desarrollar aplicaciones descentralizadas en Ethereum como se puede ver en la figura 3.4

- **Truffle**: Framework que engloba todo el ecosistema Ethereum: desarrollo de aplicaciones descentralizadas, testeo de las aplicaciones y despliegue de la misma en la Blockchain de Ethereum. Permite el desarrollo, compilación, test y despliegue de contratos inteligentes en la Blockchain. También permite el mantenimiento de redes privadas o de redes públicas de test (Rinkeby, Kovan, Ropsten). Además, este framework contiene una interfaz de línea de comandos integrada para el desarrollo de la aplicación y de scripts configurables para lanzar automáticamente los contratos a la Blockchain.[10]

- **Web3**: Librería JavaScript para comunicarse con un nodo local de la Blockchain usando el protocolo estándar de llamada de procedimiento remoto (Remote Procedue Call, RPC) sin estado. [13]

- **Solidity**: Lenguaje de programación tipado orientado a contratos inteligentes, inspirado en C++ y JavaScript. Interactúa con la Máquina Virtual Ethereum (EVM) para el compilado y despliegue de los contratos inteligentes en la Blockchain.[9]

- **Ganache/TestRPC**: Entorno para el despliegue de Blockchains privadas o locales para testing y pre-producción. Es el antiguo TestRPC, ahora en

desuso. Viene integrado con Truffle. Este entorno permite inspeccionar todo el registro de las transacciones de los bloques y el conjunto de la cadena de nuestra Blockchain privada.[4]

- **Serpent**: Lenguaje de programación orientado a contratos inteligentes inspirado en Python. Actualmente se encuentra en desuso.
- **Embark**: Framework para desarrollar e implementar aplicaciones descentralizadas sin servidor. Embark se integra actualmente con Blockchains EVM (Ethereum), depósitos descentralizados (IPFS) y plataformas de comunicación descentralizadas.[3]
- **Metamask**: Extensión del navegador que actúa como puente entre la Blockchain y el navegador. Permite visitar e interactuar con aplicaciones descentralizadas sin necesidad de ejecutar un nodo completo Ethereum.[7]

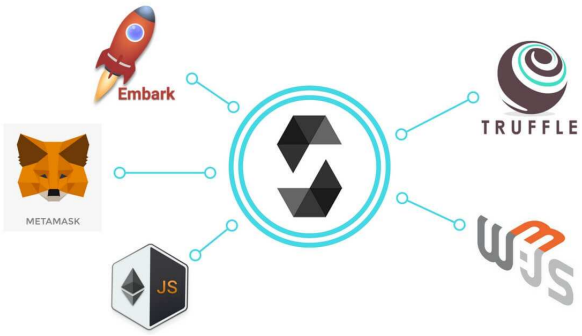


Figura 3.4: Ecosistema desarrollo aplicación descentralizada Ethereum

Capítulo 4

Diseño y desarrollo

AutoRent es una plataforma web descentralizada (DApp) [18] para el alquiler de vehículos usando contratos inteligentes en la Blockchain de Ethereum. Está desarrollada usando el Framework JavaScript Truffle, uno de los más populares en el desarrollo de aplicaciones descentralizadas. Este framework incluye todo el ciclo de desarrollo de una aplicación, pre-producción, producción, desarrollo y despliegue.

4.1. Generación de smart contracts

Truffle incluye un compilador para los contratos inteligentes desarrollados en Solidity, así como las herramientas necesarias para su despliegue en la red Ethereum *(migrations)*. Para ello hay que escribir nuestros ficheros solidity en el directorio *contracts/* con la extensión *'.sol'* para que los detecte y pueda compilarlos.

Cuenta con una Interfaz de Línea de Comandos (CLI) para facilitar esta tarea. Mediante el comando *'truffle compile'* podemos compilar los contratos. Para su despliegue en nuestra red Blockchain es necesario especificar las migraciones y luego ejecutar el comando *'truffle migrate'*. Asimismo, como se comentó en la sección anterior, al incluir todo el ciclo de vida del desarrollo, podremos lanzar nuestros test unitarios ubicados en el directorio *test/* a través de la instrucción *'truffle test'*.

4.1.1. Driving License Contract

El permiso de conducir en España coincide alfanuméricamente con el Documento Nacional de Identidad (DNI), pero para el permiso de conducir no existe una plataforma oficial del Gobierno Español que garantice su autenticidad, por lo que el arrendatario podría dar un permiso falso a la hora de alquilar los coches, la forma ideal de subsanar este fallo de seguridad sería que el Gobierno crease una Blockchain privada con los datos de todos los españoles. Al ser pri-

vada solo podrían minar transacciones el propio Gobierno pero como la cadena de bloques sería pública, pudiéndose consultar los datos.

Como alternativa, se ha creado una Blockchain privada en la que solo el administrador de la plataforma, el dueño o encargado de la empresa de alquiler de coches, pueda introducir con este contrato inteligente permisos de conducir que ha validado previamente, dotándoles de acceso para alquilar los coches.

El contrato es muy sencillo y consta de dos métodos, uno para introducir los permisos y otro para consultarlos.

4.1.2. RenACar Contract

Es el contrato inteligente encargado de los arrendamientos de la flota de vehículos. Para alquilar un coche el contrato comprueba que el permiso de circulación del cliente es correcto, es decir, que exista en la Blockchain privada de la compañía y que el coche que quiera alquilar no esté ya alquilado. Si todo es correcto, establece un mapeo entre los datos del cliente y los del coche, declarándolo como arrendatario.

Cada día que pasa, el contrato automáticamente retira de la fianza del cliente el precio diario del coche que tiene bajo alquiler. Si la fianza es insuficiente o inexistente, se le cobra un gasto extra. En cualquier momento el cliente puede añadir fondos (fianza) al contrato para prorrogar el alquiler o para que que no se le apliquen recargos.

Para que pueda devolver el coche un usuario no deberá tener cargos pendientes. De lo contrario, se rechazará la transacción y se le indicará que añada los fondos que debe. Una vez devuelto el coche, el estado del coche cambia a disponible y se elimina la relación Cliente-Coche para que otra persona puede alquilarlo. Los beneficios del alquiler del coche son enviados a la cartera digital del dueño del contrato, la persona que lo lanza a la red Ethereum. En caso de que sobrasen fondos, éstos serían enviados a la cuenta Ethereum del cliente.

4.2. Descripción

4.2.1. Arquitectura y despliegue

AutoRent ha sido desarrollada siguiendo el Modelo Vista Controlador (MVC) bajo el entorno de programación JavaScript. Para el despliegue es necesario la extensión del navegador Metamask para interactuar con un nodo Ethereum y poder ejecutar las transacciones del contrato inteligente. Con esta herramienta podemos lanzar nuestra aplicación como cualquier otra aplicación web sin tener que utilizar los navegadores propios de Ethereum como Mist.

4.2.2. Base de datos

La flota de coches es necesario guardarla en una base de datos y no en los contratos inteligentes porque ha de ser dinámica ya que se debe poder cambiar los valores de los coches, eliminar o añadir nuevos. Para la elección de la base de datos se ha optado por una NO SQL, debido a que no necesitamos persistencia de datos, es más escalable y responde mejor con JavaScript. La base de datos no relacional utilizada es MongoDB [8], una de las bases de datos NO SQL más preclaras de la comunidad

```
const Car = new Schema({
  model: String,
  price: Number,
  pricePerDay: Number,
  img: String,
  account: String
})
```

Figura 4.1: Modelo coche MongoDB

El modelo relacional de los vehículos está compuesto por un nombre del coche, un precio y precio diario en euros, un atributo imagen de la foto asociada al coche y un campo booleano para saber si el coche está alquilado o no (ver Figura **??**).

4.2.3. Paquetes NPM

JavaScript usa un gestor de paquetes node (NPM) para instalar librerías / paquetes para ser utilizados. Entre los paquetes que usa nuestra aplicación destaca:

- **Truffle Contract**: librería para interactuar con Truffle.
- **Moongose**: instancia para trabajar con MongoDB.
- **CryptoCompare**: librería para convertir de criptodivisas a divisas estándar tomando el precio en tiempo real.
- **Multer**: paquete para subir imágenes a un proceso node y guardarlas en la aplicación.
- **Web3**: API JavaScript para que nuestra aplicación descentralizada funcione en el navegador y pueda interactuar con Ethereum.

- **Vue**: librería JavaScript progresiva para el desarrollo de interfaces de usuario. [11]
- **Cron**: librería para programar tareas en el tiempo. Se usa para pasado X tiempo ejecutar una función.

4.3. Diseño

En el desarrollo del diseño de la aplicación se ha seguido un diseño sencillo, intuitivo y minimalista debido a que la plataforma está enfocada a un público general y no tienen porqué estar acostumbrados a las tecnologías de la información. Además, se ha usado el Framework Bootstrap de Twitter para conseguir un diseño responsivo, es decir, que se visualice correctamente en cualquier monitor o dispositivo.

Por otro lado, para la interacción del usuario con la plataforma se ha usado Vue siguiendo el patrón Single Page Application (SPA) donde tanto el administrador como los clientes pueden realizar cualquier tarea sin cambiar de página (ruta).

4.4. Funcionamiento

Nada más acceder a la plataforma web vemos la landing page o página principal, con el logo de la plataforma y brevemente descritas las características de esta aplicación. El diseño es bastante sencillo y minimalista como ya comentamos (ver Figura 4.2).

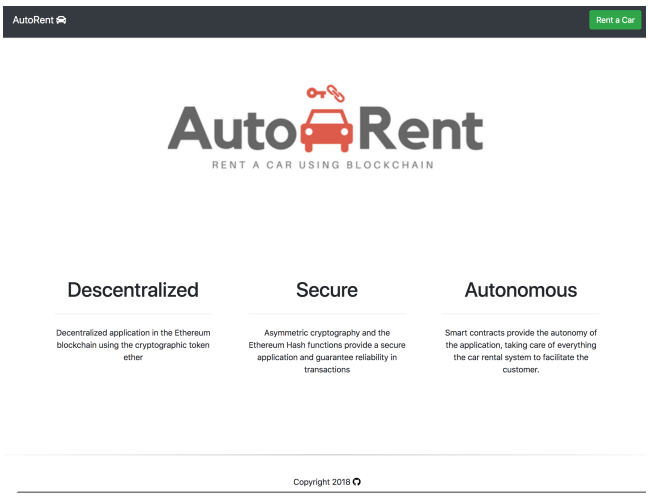


Figura 4.2: Landing page de la plataforma

Si accedemos con el botón 'Rent a Car' situado en la esquina superior derecha nos llevará a la página de la flota de vehículos, donde realizamos todas los

procesos del alquiler de coches, siguiendo el modelo SPA.

4.4.1. Flota de vehículos

En esta sección se encuentra la flota de los vehículos para su alquiler. Si el administrador inicia sesión podrá añadir nuevos coches o eliminar los que ya están en la base de datos.

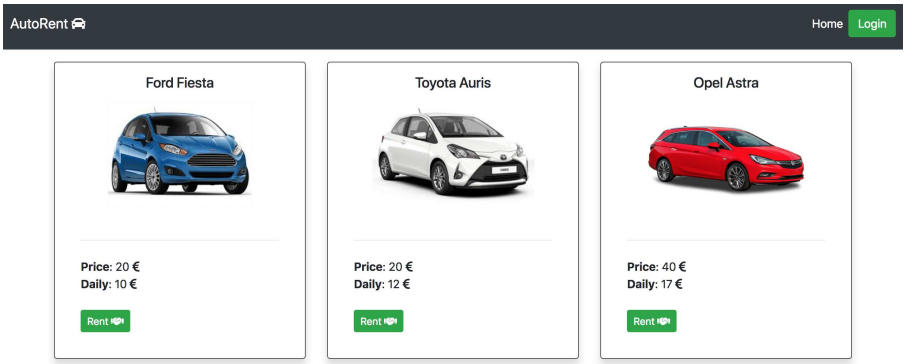


Figura 4.3: Flota de coches en la web

Si el administrador de la empresa de alquiler de vehículos inicia sesión con sus credenciales, cambiará el botón de inicio a un panel de administración (ver Figura 4.4).

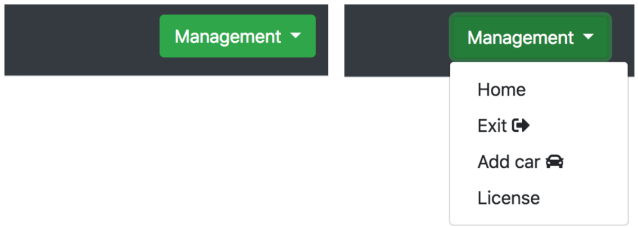


Figura 4.4: Panel de administración

Vemos en este panel que se puede volver a la página principal, cerrar sesión, añadir un nuevo coche o validar una licencia de conducir a nuestra Blockchain privada con la correspondiente transacción Ethereum.

4.4.2. Modificando la flota de vehículos

Si el administrador añade un nuevo vehículo, se le muestra un formulario para que complete este nuevo coche con sus características. Los campos corresponden con el modelo de la base de datos descrito anteriormente (ver Figura 4.5).

Por seguridad, el código solo admite ficheros acabados en .jpeg, .png, o jpg. Además el tamaño de la imagen no puede superar el megabyte.

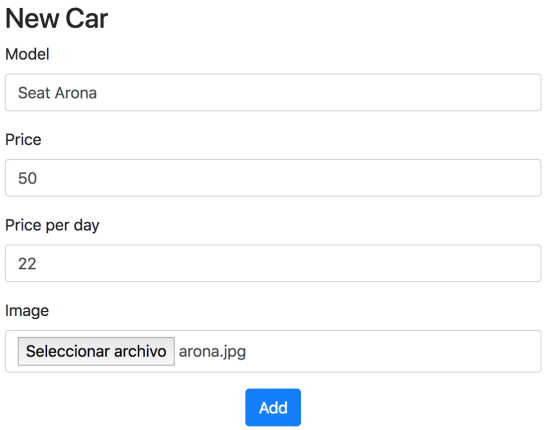


Figura 4.5: Agregando un nuevo coche

Vemos que ahora la flota de coches ha cambiado y contiene el vehículo que se acaba de añadir. Desde el perfil de administrador cada coche también contiene un botón para ser eliminado de la flota (ver Figura 4.6).

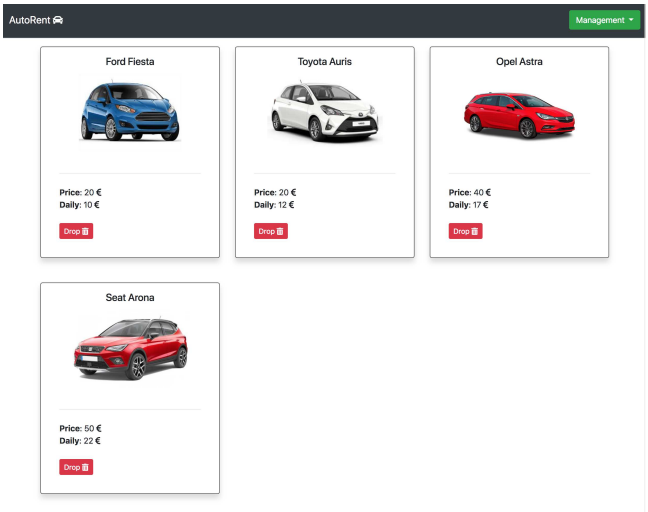


Figura 4.6: Flota de vehículos nueva

4.4.3. Transacciones: administración

Las únicas transacciones que debe hacer el administrador de la plataforma web son las de lanzar los contratos inteligentes a Ethereum. Cuando la plataforma se despliegue en un entorno real se deberá cambiar a la red principal

Ethereum y no a la de pruebas, por lo que tendrá un costo en ethers, así como las transacciones correspondientes de validación de permisos de conducir de nuestro contrato inteligente de driving license.

Si el administrador añade una licencia de conducir para que un cliente pueda alquilar, Metamask abre una ventana con la transacción (ver Figura 4.7).

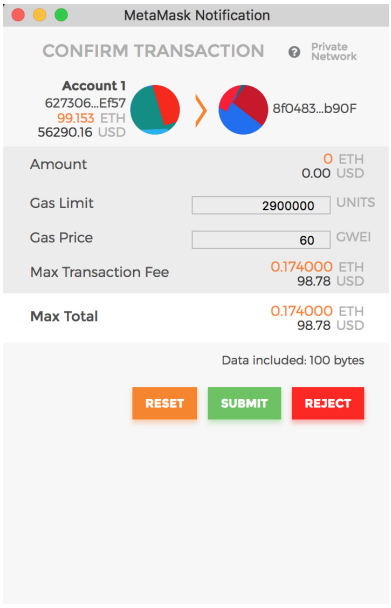


Figura 4.7: Transacción de una licencia de conducir

La transacción no es de envío de fondos, sino de ejecución de una función. Truffle distingue entre llamadas y funciones. Por un lado, una llamada no implica cambiar la información de la cadena de bloques y no cuesta dinero. Se usa para leer información de algún bloque. Por su parte, las funciones son transacciones al uso para modificar datos y sí que cuestan ethers.

Toda transacción cuesta dinero y son relativamente baratas. En este caso, es menos de un dólar. Si fuese la main net de Ethereum deberíamos gastarnos algo más de dinero para el coin base del minero.

4.4.4. Transacciones: clientes

El usuario puede alquilar, añadir fianza y devolver coches. El contrato se encarga de todas estas operaciones, así como de almacenar la fianza del usuario y cobrarle los gastos de los recargos en el caso de que el usuario no haya añadido fianza y haya pasado un día sin pagar el alquiler. El contrato también se encarga de devolver la fianza sobrante al usuario y de darle los beneficios al dueño del contrato, el administrador.

Si se selecciona un coche a alquilar, al pulsar el botón 'Rent' se mostrará en la misma ventana el formulario de información del cliente. En este formulario el usuario deberá indicar una licencia de conducir válida, es decir, que ya esté en la Blockchain (ver Figura 4.8).

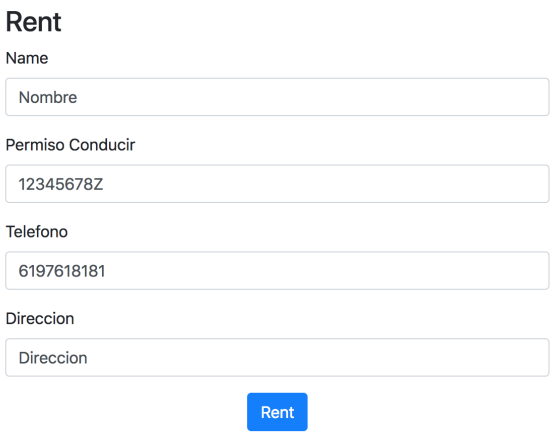


Figura 4.8: Formulario de datos del cliente

Una vez enviado el formulario se ejecuta la transacción. Podemos ver que el precio que se está enviado al contrato corresponde al "Ford Fiesta"(20 euros) que al cambio son unos 23 dólares, la moneda con la que trabaja Metamask. La transacción también incluye el precio por minarla.

Además, el cliente puede ajustar el precio que quiere pagar por la transacción modificando el gas limit y el gas price, como se explicó anteriormente (ver Figura 4.9).

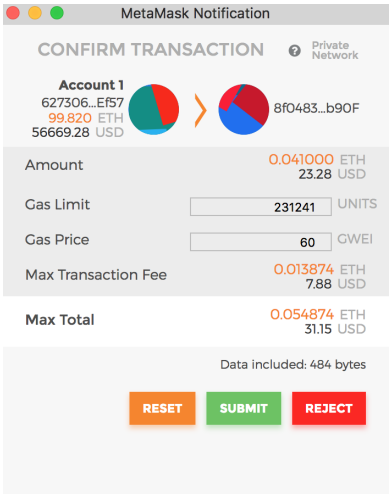


Figura 4.9: Transacción de alquiler de un coche

Una vez minada la transacción, el contrato recoge la información del cliente y del coche y las relaciona. El contrato trabaja con las direcciones hexadecimales de las cuentas de Metamask. Ahora, el coche alquilado en la página habrá cambiado tanto para el usuario que lo alquiló como para el resto de usuarios.

Las dos posibilidades para un coche alquilado son, por un lado para el usuario que ha alquilado el vehículo se le muestra las nuevas funciones. Puede devolver o añadir fianza para los cobros diarios. En otra parte, para cualquier otro usuario se le mostrará que no puede alquilarlo porque el coche ya lo está (ver Figura 4.10).

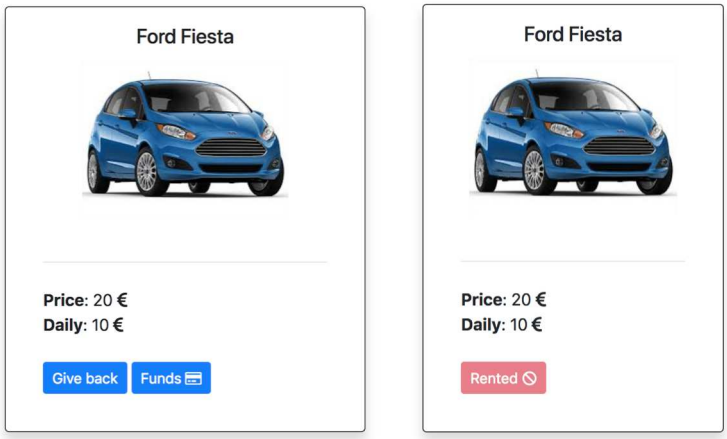


Figura 4.10: Coche alquilado en la plataforma

Como ya hemos dicho, el contrato va cobrando diariamente el precio diario de los coches alquilados para cada usuario junto a la librería Cron de JavaScript descrita anteriormente. Todos los días por lo tanto se le aplica a los clientes el cobro diario. Si el usuario en ese momento no tiene fianza suficiente o no tiene directamente, se le pasa a cobrar un recargo extra que es el propio cobro y una cantidad extra por no tener la fianza en el momento del cobro. Un usuario no puede devolver un coche si tiene recargos pendientes.

Una vez introducida una fianza superior o igual al dinero que debe el usuario, se le vuelve a habilitar el botón de devolver el vehículo. La fianza restante, si la hubiese, es devuelta al cliente y los beneficios al administrador.

4.5. Descripción detallada

La plataforma se ha desarrollado siguiendo los últimos estándares de programación JavaScript, usando ECMAScript 6 que es una especificación del lenguaje con las últimas novedades, como *async await* para los procesos asíncronos, los *import* para usar librerías externas, etc.

Para el lado del cliente como se comentó se usa Vue, uno de los framework más recientes y más usados para el Front End.

4.5.1. Despliegue contratos inteligentes

Para desplegar los contratos en la Blockchain se necesita especificar primero de todo, la Blockchain. Puede ser la Main Net, que cuesta dinero, cualquiera de las de prueba (Kovan, Ropsten, Rinkeby) o una privada como en este caso. Para hacerlo, necesitamos crear un fichero de configuración llamado 'truffle.js' para cuando se ejecute el comando 'truffle migrate' tome las configuraciones de esta función (ver Figura 4.11).

```
module.exports = {
  networks: {
    development: {
      host: '127.0.0.1',
      port: 7545,
      network_id: '*'
    }
  }
}
```

Figura 4.11: Configuración del framework Truffle

En el despliegue se especifica la red 127.0.0.1 que es nuestra red privada y el puerto de acceso. A continuación, tenemos que lanzar las migraciones para el contrato. Una migración es la operación de desplegar el contrato en la Blockchain especificada mediante un fichero javascript usando uso de la librería nativa de Truffle *artifacts* (ver Figura 4.12).

```
var Temp = artifacts.require('rentacar')

module.exports = function (deployer) {
  deployer.deploy(Temp)
}
```

Figura 4.12: Ejemplo de artifacts

Una vez desplegado en la Blockchain tenemos que interactuar con él. La migración nos convierte el contrato escrito en solidity a un fichero .json para reducir el tamaño del mismo.

Finalmente, como Metamask es una extensión del navegador y es la que interactúa con los nodos Ethereum, tenemos que trabajar con el contrato desde el lado del navegador. Para ello antes de cargar la página de Flota de Vehículos con la propiedad Vue *beforeCreate* cargamos esos ficheros json y le especificamos el proveedor http, que debe ser el mismo que Metamask (ver Figura 4.13).

```
axios.get('drivingLicense.json').then(response => {
  myContract2.contracts.Contract = TruffleContract(response.data)
  myContract2.contracts.Contract.setProvider(myContract.web3Provider)
```

Figura 4.13: Despliegue de un contrato inteligente

4.5.2. Funciones relevantes de los contratos inteligentes

En estado apartado se detallan en profundidad algunas funciones relevantes del contrato inteligente encargado de todo el proceso de alquiler de un vehículo.

Por un lado, la función *AlquilarCoche* recibe todos los parámetros del cliente y del vehículo a alquilar. Esta función comprueba que el coche no esté alquilado y que la licencia que ha recibido por parámetro exista en el otro contrato, encargado de las mismas. Se requiere especificar la dirección hexadecimal única donde está desplegado dicho contrato ya que, como hemos comentado cada contrato representa una dirección en la Blockchain y necesitan comunicarse.

Una vez comprobadas las condiciones de arrendamiento se hace un mapeo Coche-Cliente usando la dirección de la cuenta Metamask del cliente y asociándola al coche que ha alquilado (ver Figura 4.14).

```
function alquilarCoche(string _usuario,string _direccion, string _nPermiso, uint _telefono, string _
    drivingLicense permiso = drivingLicense(0x345ca3e014aaf5dca488057592ee47305d9b3e10);
    require(permiso.getLicense(_nPermiso));
    require(!coches[_id].alquilado);
    coches[_id] = Coche({
        precioCoche: _precioCoche,
        precioDiario: _precioDiario,
        alquilado: true
    });
    clientes[msg.sender] = Cliente({
        usuario: _usuario,
        direccion: _direccion,
        nPermiso: _nPermiso,
        telefono: _telefono,
        fianza: 0,
        recargos: 0
    });
}
```

Figura 4.14: Función alquiler de vehículos

En cuanto a la función *devolver*, ésta comprueba que el cliente no tenga cargos pendientes. En el caso de que sea falsa la comprobación devuelve la fianza que le ha sobrado al cliente.

Asimismo, también envía al dueño del contrato el resto de los beneficios y marca el coche como disponible para que pueda volver a ser alquilado por otro cliente (ver Figura 4.15).

```
function devolverCoche(string _id) {
    require(clientes[msg.sender].recargos == 0);
    msg.sender.send(clientes[msg.sender].fianza - ownerBenefits);
    owner.send(clientes[msg.sender].fianza);
    coches[_id].alquilado = false;
    clientes[msg.sender].fianza = 0;
    owner.send(this.balance);
    ownerBenefits = 0;
}
```

Figura 4.15: Función devolver vehículos

4.5.3. Diferencia entre Truffle Transaction y Truffle Call

Una transacción Truffle es cualquier procedimiento para modificar la información del contrato. Esta operación consume gas, por lo que son necesarios ethers para minarla (ver Figura 4.16).

```
backCar: async function(identificador) {

  id_ = identificador
  account = web3.eth.defaultAccount
  self = this
  let instance = await myContract.contracts.Contract.deployed()
  await instance.devolverCoche(id_, {from: account, gas:2900000})
  axios.post('/car/'+id_+'', { account: '' }).then(function (response) {})
  location.reload()
```

Figura 4.16: Transacción framework Truffle

En cambio, una llamada no requiere de gas porque es simplemente leer un dato. Por ello, no se considera una transacción al uso y es instantánea puesto que no hace falta minarla (ver Figura 4.17).

```
getDebt: async function() {
  self = this
  account = web3.eth.defaultAccount
  let instance = await myContract.contracts.Contract.deployed()
  let recargos = await  instance.getRecargos.call({from: account})
  let value = web3.fromWei(recargos.toString(), "ether")
  await $.get('/convertReverse/'+value+'', function(data) {
    self.$store.commit("changeDebt", parseInt(data.price))
    location.reload()
  })
```

Figura 4.17: Llamada framework Truffle

4.5.4. Vue y Vuex

Vue mejora la librería jQuery y hace las operaciones en el cliente mucho menos costosas, simplificando el código en número de líneas. Vuex [12] es una librería complementaria para Vue siguiendo el patrón Flux para centralizar el estado de la información, lo que permite que cuando cambie algún valor no se tenga que replicar entre los componentes de Vue.

En nuestro caso se usa para centralizar el precio que deben los clientes. Este atributo además tiene la propiedad computada de Vue, es decir, que si cambia el valor de la variable *débito* automáticamente se actualiza, sin necesidad de refrescar la ventana.

```
var store = new Vuex.Store({
    state: {
        debt: ''
    },
    getters: {
        getDebt: state => {
            return state.debt;
        }
    },
    mutations: {
      changeDebt: (state,debt) => {
        state.debt = debt
    },
    }
  })
```

Figura 4.18: Ejemplo estados de Vuex

Asimismo, la información es almacenada en un *vector store* que contiene los atributos *getters*, *state* y *mutations (setters)*. El *state* es el estado inicial y las *mutations* se usan para modificar el *state* (ver Figura 4.18).

4.5.5. Cron

Cron es la alternativa NPM para el framework Ethereum Alarm Clock descrito anteriormente, y es el encargado de ejecutar cada X tiempo las operaciones que le indiquemos. Simplemente hay que especificar la zona horaria de referencia para las funciones de tiempo y cada cuanto queremos que se ejecute esa función en el tiempo. Puede ser segundos, minutos, horas, días de la semana, días del mes o meses.

En AutoRent se usa para cobrar diariamente a todos los clientes con un coche en alquiler (ver Figura 4.19).

```
const CronJob = require('cron').CronJob;
import {dailyCharge} from '../app/controllers/mainController'
function schedule() {
    new CronJob('0 0 0 * * *', function() {
        dailyCharge()
    }, null, true, 'Atlantic/Canary');
}

module.exports = { schedule }
```

Figura 4.19: Configuración paquete Cron

4.5.6. Cryptocompare

CryptoCompare es otro paquete NPM y a su vez la API de la plataforma de criptodivisas del mismo nombre. Sirve para listar todos los tokens criptográficos y permite hacer conversiones entre ellos o a otras monedas físicas como euros o dólares.

En nuestro proyecto se usa para pasar de euros, que es la moneda que trata con el cliente, a ethers, el token que entiende nuestros contratos inteligentes.

```
async function convert (value) {
    let price = await cc.price('EUR', ['ETH'])
    let resultado = Object.values(price) * value
    return resultado
}

async function convertReverse (value) {
    let price = await cc.price('ETH', ['EUR'])
    let resultado = Object.values(price) * value
    return resultado
}
```

Figura 4.20: Cambio de criptodivisas con CryptoCompare

Capítulo 5

Pruebas y resultados

Apartado para presentar los problemas surgidos durante el desarrollo del proyecto, como los resultados tanto teóricos (ventajas y desventajas) como a nivel práctico (pruebas).

5.1. Problemas en el proceso de desarrollo

A continuación se detallan los principales problemas surgidos durante el proceso de desarrollo de la aplicación descentralizada.

5.1.1. Imágenes MongoDB

Cada coche de la flota de la empresa de alquiler tiene que tener una imagen para que el cliente vea de manera visual el vehículo. Para ello se debe almacenar en la entidad 'Coche' de la base de datos en MongoDB un atributo imagen.

La primera aproximación fue guardar directamente la imagen en la base de datos pero no se pudo almacenar directamente porque la base de datos crecería de manera exponencial y no sería una buena práctica. De esta manera, se decidió codificar la imagen en base64 para hacerla más ligera, descodificándola luego para mostrarla. El problema de este proceso es que es tedioso y no muy óptimo, por lo que finalmente se planteó guardar el atributo Imagen como una string que apuntase a la ruta donde se encontraba esa imagen en el proyecto de la aplicación. El administrador cuando crea un nuevo coche en la flota, sube la imagen y esta es guardada en la ruta *'/public/images/cars/idcoche.png'*, cuando se introduce la entidad coche en la base de datos, el atributo contiene la ruta exacta de su imagen correspondiente.

5.1.2. Interacción Ethereum - Navegador

Las transacciones de Ethereum necesitan conectarse a un nodo para ser minadas. Si queremos que nuestra aplicación esté en la web necesitaremos un puente

entre Ethereum y nuestro navegador. Para ello usamos la herramienta Metamask. Metamask usa la librería JavaScript Web3 para lanzar las transacciones. Esta librería necesita estar en el front-end de nuestra aplicación para que se pueda comunicar con Metamask.

Se tuvo que rotar parte de la lógica de negocio (control de transacciones del contrato) del back-end al front-end, usando ahora el framework progresivo Vue JavaScript para facilitar el desarrollo en el cliente y no usar JavaScript puro o JQuery, que son más redundantes e incrementan la complejidad de nuestro código.

5.1.3. Ethereum Alarm Clock

Ethereum Alarm Clock es una librería Ethereum para ejecutar transacciones programadas en el tiempo. Estas transacciones debían ser minadas por la comunidad de EAC. Por ejemplo, una persona concretaba que X transacción debía ser minada a las Y horas de un día concreto y pagaba un costo para que una persona de la comunidad la minase. Sin embargo, la comunidad de mineros desapareció y el proyecto quedó obsoleto.

Otras de las alternativas para ejecutar una función automáticamente programada en el tiempo era Oraclize, librería Oracle para Ethereum. El problema es que resultaba muy costosa en términos de gas para la empresa ejecutar una transacción para todos los clientes cada día.

Estas dos comunidades eran las más activas pero no satisfacían nuestras necesidades. Además, a día de hoy no existe una alternativa real a esto. Por ello, se decidió usar una librería JavaScript para programar que todos los días a todos los clientes en activo (que tienen un coche en alquiler) se les cobre el precio correspondiente de cada coche.

5.2. Testing

Como ya se comentó, Truffle incluye por defecto todo el ciclo de vida de desarrollo de una aplicación, incluidas las pruebas o el testing. Las pruebas cubren todas las funciones de los dos contratos.

- **Test Driving License**
  Se comprueba que se puedan insertar permisos de conducir correctamente, es decir, una vez introducido que podamos obtener el permiso y previamente establecido. También se comprueba que no se puede pedir un permiso de conducir que no se ha introducido.

- **Test Rent a Car**
  Se valida que funcione correctamente cualquier proceso del sistema de alquiler de un coche. No se puede alquilar un coche si el dni no ha sido

introducido previamente en la Blockchain, y se pueden alquilar los que sí han sido introducido y validados. No se puede alquilar un coche que ya está alquilado y sí lo que están no alquilados. Se puede introducir, devolver y obtener la fianza de un cliente correctamente. Se comprueba que se cobra diariamente el precio correspondiente y que se pueden obtener los recargos. No puede devolver un cliente un coche que tiene recargos y sí puede hacerlo si no los tiene.

```
trojan at NS in ~/Desktop/TFG on master*
[$ truffle test
Using network 'development'.


  Contract: Driving License Test
    ✓ Se pueden insertar permisos de conducir correctamente (108ms)
    ✓ Se comprueba que no se puede pedir un permiso de conducir que no se ha introducido previamente

  Contract: Rent A Car Test
    ✓ No se pueden alquilar coches sin un dni previamente validado
    ✓ Se pueden alquilar coches con un dni validado (64ms)
    ✓ No se pueden alquilar un coche que ya esta alquilado
    ✓ Se puede introducir fianza correctamente
    ✓ Se puede obtener fianza correctamente
    ✓ Se puede obtener recargos correctamente
    ✓ Funcion cobro diario (61ms)
    ✓ Funcion cobro diario con recargos (66ms)
    ✓ No se pueden devolver coches con recargos
    ✓ Se pueden devolver coches sin recargos (121ms)
    ✓ Devolver un coche sin cobros, devuelve la fianza integra (406ms)
    ✓ Devolver un coche con cobros, no devuelve la fianza integra (459ms)


  14 passing (2s)
```

Figura 5.1: Test unitarios de Truffle

Los test unitarios con Truffle siguen la metodología estándar de otros frameworks de pruebas, cada función a comprobar es abstraída en un bloque de código o *'it'* y se espera que el resultado hipotético y el real coincidan.

Para la fianza, por ejemplo, se comprueba que la transacción se haya ejecutado correctamente. El usuario envía 1 ether al contrato (ver Figura 5.2).

```
it("Se puede introducir fianza correctamente", async() => {
    let instance = await RentACar.deployed()
    let transaction = await  instance.depositarFianza({value: web3.toWei(1, "ether"), from: account})
    let expected = 1
    assert.equal(expected, transaction.receipt.status)
})
```

Figura 5.2: Unit test de la fianza

En otro prueba unitaria se comprueba que realmente tiene esa fianza, es decir, se comprueba que el resultado de llamar a la función *'Obtener fianza'*, es decir,

el valor esperado, sea el mismo que el valor real, que en este caso es 1 ether. (ver Figura 5.3).

```
it("Se puede obtener fianza correctamente", async() => {
    let instance = await RentACar.deployed()
    let call = await  instance.getFianza.call({from: account})
    let expected = 1000000000000000000 // 1 Ether en Wei
    assert.equal(expected, call.toString())
})
```

Figura 5.3: Unit test de obtener la fianza

5.3. Ventajas AutoRent

Se pueden catalogar las ventajas de en dos tipos, ventajas frente a aplicaciones tradicionales y ventajas frente a otras aplicaciones descentralizadas.

5.3.1. Aplicaciones tradicionales

La lógica de negocio recae en los contratos inteligentes, y por tanto no se precisa de registros, autenticación y control de sesiones. Haciendo la aplicación más fácil de usar e intuitiva, el cliente con su billetera digital de Ethereum puede realizar cualquier transacción en la aplicación. Se prescinde también de una Pasarela de Pago (Terminal Punto de Venta Virtual) por lo que se ahorran los costos relacionados con un TPV o la complejidad en la plataforma al no tener que diseñar una.

5.3.2. Otras aplicaciones descentralizadas

Las principales aplicaciones descentralizadas usan su propio token criptográfico. En cambio AutoRent usa el token establecido por Ethereum, el ether. Así la empresa no tiene que emitir al mercado tokens y para el usuario final es más fácil comprar ether que una criptodivisa en particular.

Además, Darenta, Helbiz y las otras compañías intentan crear una comunidad de redes de clientes para compartir los vehículos particulares, mientras que, AutoRent define la flota de coches con un precio estable estipulado por la empresa, es decir, estas empresas están orientadas a redes sociales como AirBnB y AutoRent es la plataforma prototipo de una empresa de alquiler de coches.

5.4. Seguridad

- Los usuarios no se registran en ninguna base de datos por lo que su información no es vulnerable a ataques de seguridad. Su información la gestiona el contrato inteligente.
- Las transacciones no están centralizadas en una TPV Virtual, vulnerables a ataques de seguridad para robar las criptodivisas. Cualquier token criptográfico es almacenado en el contrato inteligente, desplegado en el Blockchain, dificultando el robo de divisas.
- Las aplicaciones descentralizadas son más robustas que las tradicionales ya que en el caso de que se quisiera atacar una de estas no bastaría con tumbar la plataforma, sino también acceder y alterar la Blockchain.
- La integridad de Blockchain hace prácticamente imposible alterar malintencionadamente cualquier dato de una aplicación descentralizada que esté alojado en ella.
- La descentralización de la cadena de bloques provoca que cualquier ataque de denegación de servicios (distributed denial-of-service attack, DDoS) sea computacionalmente muy difícil, ya que tirar un nodo no cambiaría nada y habría que tirar todos los nodos menos uno para que dejara ser una red distribuida, por lo que aplicaciones que usen esta red están menos expuestas a estos ataques.

Capítulo 6

Presupuesto

A continuación, se muestra el presupuesto estimado del desarrollo de este Trabajo de Fin de Grado junto con el despliegue de la aplicación descentralizada en Ethereum

6.1. Presupuesto personal

Presupuesto relacionado con los costes humanos, horas de trabajo dedicadas a la documentación y aprendizaje de nuevas tecnologías e implementación de estas.

Recurso	**Horas**	**Precio Unidad**	**Total**
Estudio Tecnología Bitcoin	35	12 €	420 €
Estudio Tecnología Ethereum	35	12 €	420 €
Estudio Smart Contracts	10	12 €	120 €
Tutorial Truffle	5	12 €	60 €
Estudio DApps	15	12 €	180 €
Diseño App	15	12 €	180 €
Ethereum Alarm Clock	30	12 €	360 €
Bases de Datos	25	12 €	300 €
Web3	12	12 €	144 €
Ganache	8	12 €	96 €
Front End	50	12 €	600 €
Back End	65	12 €	780 €
Total	305	-	3660 €

Tabla 6.1: Tabla resumen del presupuesto personal

6.2. Presupuesto componentes

Presupuesto donde se recoge el equipamiento, tanto físico como lógico, y la puesta en funcionamiento de un aplicación descentralizada en la red Ethereum.

Recurso	**Cantidad**	**Precio Unitario**	**Total**
Ordenador	1	750 €	750 €
Despliegue Blockchain	1	100 €	100 €
Dominio y Servidor	1	20 €/ año	20 €/ año
Total	-	-	870 €

Tabla 6.2: Tabla resumen del presupuesto de componentes

6.3. Coste total

Importe final estimado, recogiendo los presupuestos anteriores.

Presupuesto	**Costo**
Personal	3660 €
Componentes	870 €
Total	4530 €

Tabla 6.3: Tabla resumen del costo total

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones del desarrollo del presente Trabajo Fin de Grado y las posibles implementaciones futuras de cara a seguir trabajando en la línea del alquiler de coches usando Blockchain.

7.1. Conclusiones

AutoRent ha sido desarrollada siguiendo los estándares de una aplicación descentralizada. Sus principales características son:

- Es de código abierto y ninguna entidad controla los tokens.
- Todos los datos del alquiler de coches y los clientes son almacenados en una Blockchain pública y descentralizada.
- La aplicación usa un token criptográfico (ether).
- Usa el algoritmo de Prueba de Trabajo para sus transacciones.

Estas directrices nos confieren características diferentes a las tradicionales aplicaciones web, el usuario no tiene que registrarse, no se almacenan contraseñas ni se controlan las sesiones del usuario. El contrato inteligente es el encargado de usar la billetera digital del cliente y almacenar los datos en el mismo.

El contrato inteligente dota de autonomía y de inteligencia a la aplicación, pues es el encargado del alquiler y retorno de coches, almacenar el dinero, devolverlo, repartirlo y cobrar automáticamente a todos los clientes diariamente, evitando herramientas de cobros de tercero.

7.2. Líneas futuras

En los próximos años el desarrollo de las aplicaciones se asentará de manera definitiva. El Internet de las Cosas, la Inteligencia Artificial y el Blockchain dotarán las aplicaciones inteligentes de autonomía y de capacidad para gestionar y tomar decisiones de manera autónoma.

La línea actual del presente trabajo permite avanzar en estos campos mencionados y dotar de las siguientes características a AutoRent, añadiéndole valor y funcionalidades.

- Una Blockchain pública con todos los datos de los ciudadanos españoles, permiso de conducir, DNI, localización, etc. Cualquier aplicación descentralizada podrá consumir los datos de esta Blockchain para sus contratos inteligentes y desaparecerán la mayoría de los registros y bases de datos de usuarios ya que la autenticación y verificación del usuario tendrá lugar en dichos contratos. Implicará la eliminación del contrato inteligente de los permisos de conducir que tiene la empresa para validarlos, así como una reducción de complejidad.

- Con el Internet de las Cosas y la Inteligencia Artificial la flota de coches contará con sensores geolocalizadores para conocer el punto geográfico en todo momento del mismo.

- Los coches contarán con llaves digitales que les transferirá el contrato en el momento de la transacción del alquiler. Así, el cliente no tendrá que ir a punto físico a recoger las llaves. Esto permitirá que pueda dejar el coche en cualquier punto y, al estar geolocalizado, el próximo cliente pueda dirigirse a dicho lugar para abrirlo con su llave digital correspondiente.

- Las criptomonedas fluctúan demasiado su valor y, el precio de los coches, está sujeto a estos tokens. Se podrá emitir un token propio con valores estables para hacer más fácil la comprensión de precios por parte de los clientes.

- Estudiar el nuevo Reglamento General de Protección de Datos (GDPR) y el impacto que tendrá en las Blockchain, tanto en las públicas como en las privadas. Realizar los cambios si fuese necesario para cumplir con la nueva ley.

Capítulo 8

Conclusions and future works

The conclusions of the development of the present Final Degree Project are detailed and the possible implementations to face to keep working in the business of car rental service using Blockchain

8.1. Conclusions

AutoRent has been developed following the standards of a decentralized application:

- It's open source and none enterprise control the tokens.
- All data of rental car service and customers are stored in a public and in an decentralized Blockchain.
- The application uses a cryptographic token (Ether).
- Uses the Work Proof Algorithm for their transactions.

These guidelines give us different characteristics to traditional applications web. The user doesn't need to register, doesn't store password and there is no control over the user's sessions. The Smart Contract is in charge to use customer's digital wallet and store the data in the same one.

The Smart Contract gives to the application autonomy and intelligence, because he's in charge of the renting and returning of the vehicles, storing and returning the money, distributing and charging automatically to each customer every day, avoiding other ways of payment.

8.2. Future Works

Internet of Things, Artificial Intelligence and Blockchain will definitely settle in the applications development in the next years. Will be smart applications with autonomy and capacity to manage and take decisions autonomously.

The current line of work allows us to advance in these fields and give these characteristics to AutoRent, giving valor and functionality.

- A public Blockchain with all data of the spanish citizens (Driving license, ID, location, etc). Any decentralized application may use the data of this Blockchain for his smart contracts and most of these registries and databases will disappear because the user's authentication and verification will be made in the contracts. It will imply the elimination of the driving license smart contract that the company has to validate them, as well as a reduction in complexity.

- With Internet of Things and Artificial Intelligence the net of vehicles will have geolocation sensors to know the exact location of each vehicle in every moment.

- The vehicles will have digital keys that will be transferred by the contract in the transaction moment. So, the customer no need going to a physical point to take the keys. This will allow that the customer can left the car where ever he wants and the next customer can go to those point because the car is geolocated.

- The value of the cryptocurrency fluctuates a lot and obviously the price of the vehicles. It will be possible to issue your own token with a stable value to make easy the politicy of prices compression.

- To study the new General Data Protection Regulation (GDPR) and the impact it will have on the Blockchain, both in publics and privates. Make the changes if necessary to comply with the new law.

Bibliografía

[1] Blockchain Demo. https://anders.com/blockchain/. Último acceso 2018-05-29.

[2] Darenta. https://darenta.io/. Último acceso 2018-05-28.

[3] Embark 2.5.2 documentation. https://embark.readthedocs.io/en/2.6.6/. Último acceso 2018-05-28.

[4] Ganache — Truffle Suite. http://trufflefframework.com/ganache/. Último acceso 2018-05-28.

[5] Helbiz. https://helbiz.com/. Último acceso 2018-05-28.

[6] HireGo. https://hirego.io/. Último acceso 2018-05-28.

[7] MetaMask. https://metamask.io/. Último acceso 2018-05-28.

[8] MongoDB. https://www.mongodb.com/. Último acceso 2018-05-28.

[9] Solidity. http://solidity.readthedocs.io/en/v0.4.21/. Último acceso 2018-05-28.

[10] Truffle Suite. http://trufflefframework.com/. Último acceso 2018-05-28.

[11] Vue Js. http://vuejs.org/. Último acceso 2018-05-28.

[12] Vuex. https://vuex.vuejs.org/. Último acceso 2018-05-28.

[13] web3.js. http://web3js.readthedocs.io/en/1.0/index.html. Último acceso 2018-05-28.

[14] M. Antonopoulos. Andreas. *Mastering Bitcoin*. Dic 2014.

[15] Antonio Marín Bermúdez. *Estudio de la utilización de protocolo blockchain en sistemas de votación electrónica*. PhD thesis, Universitat Politècnica de Cataluña, 2016.

[16] ConsenSys. A 101 noob intro to programming smart contracts on ethereum. *Medium*, 2015.

[17] Henry Cuttell. *Blockchain-based Smart Tenancy Agreements*. PhD thesis, Imperial College London, 2017.

[18] Merunas Grincalaitis. The ultimate end-to-end tutorial to create and deploy a fully decentralized dapp in ethereum. *Medium*, 2017.

[19] Satoshi Nakamoto. Bitcoin: Un Sistema de Efectivo Electrónico Usuario-a-Usuario. www.bitcoin.org.