Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

IMPLEMENTACIÓN DE LA TECNOLOGÍA BLOCKCHAIN EN LA GESTIÓN DEL NEGOCIO DE ALQUILER DE VEHÍCULOS

Autor: Pablo Ruiz Giles

Tutor: Dr. Juan Ignacio Pérez Sanz

Cotutor: Dr. Raúl Durán Díaz

2020-2021

ESCUELA POLITECNICA
SUPERIOR

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

IMPLEMENTACIÓN DE LA TECNOLOGÍA BLOCKCHAIN EN LA GESTIÓN DEL NEGOCIO DE ALQUILER DE VEHÍCULOS

Autor: Pablo Ruiz Giles

Tutor: Dr. Juan Ignacio Pérez Sanz

Cotutor: Dr. Raúl Durán Díaz

TRIBUNAL:

Presidente: Rafael Rico López.

Vocal 1º: Pablo Muñoz Martínez.

Vocal 2º: Juan Ignacio Pérez Sanz.

FECHA: 14 de julio de 2021

CALIFICACIÓN: ……

ESCUELA POLITECNICA
SUPERIOR

Resumen

Este trabajo implementa un portal web para una compañía de alquiler de vehículos, que permite tanto la gestión empresarial como el uso de sus servicios por parte de los clientes.

Como aspecto novedoso, el portal web utiliza "Blockchain" como tecnología base. Una de las propiedades características de esta tecnología es que permite la gestión descentralizada de transacciones sin que sea necesaria la intervención de una tercera parte de confianza.

En particular, se utilizará "Ethereum" que hace posible los llamados "contratos inteligentes" (smart contracts). Aprovechando sus capacidades, la plataforma dará cumplimiento a los requisitos que típicamente se piden a un portal que preste un servicio como el alquiler de vehículos. Además, la aplicación puede trasladarse de manera casi directa a una red real.

En su implementación, se ha hecho uso del conjunto de aquellas herramientas de diseño y desarrollo presentes en el mercado que se han considerado más convenientes.

Palabras clave:

Blockchain, vehículo, Solidity, Ethereum, Smart Contract.

ESCUELA POLITECNICA
SUPERIOR

Abstract

This paper implements a web portal for a car rental company that enables both business management and the use of its services by customers.

As a novel aspect, the web portal uses "Blockchain" as a base technology. One of the characteristic properties of this technology is that it enables the decentralized management of transactions without requiring the intervention of a trusted third party.

In particular, "Ethereum" will be used which makes so-called "smart contracts" (smart contracts) possible. Leveraging its capabilities the platform will fulfill the requirements typically asked of a portal providing a service such as car rental. In addition, the application can be almost directly transferred to a real network.

In its implementation, use has been made of the set of design and development tools present in the market that have been considered the most convenient.

Keywords:

Blockchain, vehículo, Solidity, Ethereum, Smart Contract.

ESCUELA POLITECNICA
SUPERIOR

Índice

CAPÍTULO 1

Introducción al proyecto

1. Introducción

La irrupción de una nueva tecnología como el Blockchain, que se encuentra en plena fase de desarrollo y expansión, presenta la oportunidad de aplicarla en ámbitos de empresas e instituciones para administrarlas de forma distinta a la tradicional.

Esta tecnología supone un paso definitivo a otra forma de gestionar los negocios, debido a las características de seguridad que aporta en la administración y el tratamiento de datos, así como en la descentralización de su gestión, con la optimización de los recursos humanos que ello conlleva.

Con el presente proyecto se ha buscado aplicar las características que presenta el Blockchain a un modelo de negocio en concreto como es el *rent a car*, donde su implementación aportará la descentralización en la gestión de operaciones y transacciones, aumento de seguridad en el tratamiento y almacenamiento de datos, mayor fluidez y la eliminación de intermediarios. Todo ello repercutirá en ahorro de costes considerable que lo hará muy atractivo a las empresas del sector.

1.1. Presentación del estudio.

El mercado del *rent a car* en Europa está en crecimiento y los datos para el futuro a medio y largo plazo son muy optimistas.

Estos modelos de negocio utilizan, de manera estratégica, las herramientas de tecnología informática que les permiten automatizar muchos de sus procesos, lo que les supone una reducción considerable en sus costes.

Además, los softwares de gestión les permiten conocer a su público objetivo y almacenar una gran cantidad de información para estudiar detenidamente sus verdaderas necesidades y adaptarse con rapidez a ellas:

- Aprovechan las herramientas de la tecnología informática.
- Se mantienen en constantes actualizaciones.
- Han logrado un sentimiento de confianza en el consumidor.
- Utilizan plataformas digitales para entornos móviles.
- Invierten en el control y el mejoramiento constante de su reputación online.
- Ofrecen experiencias que satisfacen necesidades.

Las empresas de alquiler de vehículos deben intentar ofrecer una flota de vehículos variada, de manera que los clientes encuentren un medio de transporte que se adapte a sus necesidades. Los negocios, que ofrecen este tipo de vehículos, los podemos clasificar en los siguientes grupos: económicos, medio, ejecutivo, lujo, monovolumen, minibús, furgoneta y motocicleta/ciclomotor.

	Económicos	**Medio**	**Ejecutivo**	**Lujo**	**Monovolumen**	**Minibús**	**Furgoneta**	**Motocicleta**
Puertas	3/5	5	5	5	7	10	3/5	
Ocupantes	4	5	5	5	5	5	5	½
Consumo	Bajo	Medio	Alto	Alto	Medio	Medio	Medio	Bajo

Tabla 1: Tipos de vehículos.

Todo lo descrito anteriormente conlleva un proceso laborioso de administración y gestión, por su gran volumen y variedad, al que las empresas dedican actualmente una parte muy importante de sus recursos humanos, materiales, financieros y tecnológicos, con el sobrecoste que todo ello conlleva.

1.2. Objetivos del proyecto.

Este TFG tiene por objetivo el estudio y el análisis de la implementación, mediante Blockchain, de la gestión de recursos en las empresas del *rent a car*. Se estudiarán las ventajas que puede presentar, así como sus fortalezas y debilidades. Los objetivos perseguidos son:

- El cliente podrá alquilar el/los vehículos ofrecidos por las empresas de *rent a car* en cualquier punto del espacio europeo y verificar de forma inmediata su disponibilidad y localización. Además, este sistema de gestión permitirá que el cliente retire o entregue el vehículo en cualquier oficina de alquiler (*sucursal*) o ubicación señalada a tal efecto por la

empresa independientemente de donde esté situada, encontrándose el mismo localizado por el sistema de forma inmediata, lo que facilita su nuevo alquiler.

- Se plantea trabajar en formas de fusionar IoT con los sistemas de Blockchain, donde un contrato inteligente (*Smart Contract*) podrá verificar que realmente el vehículo que se pretende alquilar se encuentra en el lugar fijado, está disponible y permitirá fijar los términos en los que se realizará el alquiler. El contrato también facilitará el pago, a través de su Wallet (*monedero electrónico asociado a una cuenta de Blockchain, en nuestro caso Ethereum*). Teniendo su utilización la ventaja del ahorro de costes derivados de las transacciones financieras, como son por ejemplo los honorarios y comisiones que se pagan a bancos y tarjetas de crédito.

- Para las compañías aseguradoras el uso del Blockchain junto con los Smart Contracts, aplicado a las flotas de alquiler de vehículos, podría facilitar nuevos tipos de seguro. Ahora se pagan cantidades poco personalizadas y, gracias a la implementación de esta tecnología, podrían ofrecer un seguro basado no sólo en la edad o la experiencia, sino también en el uso específico del vehículo (gracias a la trazabilidad). En la implicación de responsabilidades del conductor el uso de Blockchain permitirá saber exactamente el momento en el cual comienza el alquiler del vehículo y su finalización, siendo guardadas junto a sus las coordenadas geográficas. Esto junto a los contratos inteligentes, también permite activar y desactivar las reclamaciones automatizadas según trazabilidad y eventos en el vehículo.

- Un aspecto importante del uso de Blockchain es la competitividad y el ahorro de costes en la gestión de flotas. Hoy día, las redes de *rent a car* están compuestas por numerosas oficinas de alquiler, operadas y gestionadas de forma individual, donde sus estructuras administrativas (de personal y gestión) están replicadas en todas por igual. Debido a esta centralización, la redundancia de recursos supone un enorme coste de administración y gestión de todas las flotas. Por tanto, se propone como solución la adopción de un sistema de gestión descentralizado, donde es posible optimizar los recursos humanos en todos los ámbitos de la empresa, que repercutirá un ahorro de costes.

- Para poder gestionar la aplicación por parte de todas las delegaciones de alquiler de la compañía, se diseñará un interfaz gráfico que puede ser accedido y gestionado directamente, tanto por los administradores de las distintas sucursales como por los clientes. Todos ellos podrán utilizarla accediendo a través del uso de teléfonos móviles, tablets, portátiles y cualquier PC instalado en el ámbito doméstico o empresarial.

1.3. Trabajo a Desarrollar.

Para desarrollar el trabajo que se ha propuesto, se dividirá y estructurará en capítulos, abordando en cada uno un apartado del proyecto:

- Capítulo 2: en este capítulo nos centraremos en indagar sobre qué entendemos por Blockchain y la tecnología que lo sustenta, estudiando todos los elementos que intervienen para poder hacer posible su funcionamiento.

- Capítulo 3: se abordará la red Ethereum, se explicará el funcionamiento de la misma en profundidad y el uso de los Smart Contracts, Wallets y otros elementos que son necesarios para el desarrollo del proyecto.

- Capítulo 4: se empleará para desarrollar las tecnologías que intervendrán a lo largo del proyecto y, por ende, necesarias para poder tener una comprensión completa de todos los elementos que intervienen en él.

- Capítulo 5: contiene el diseño del proyecto así como las interacciones necesarias entre las tecnologías que confluyen en él.

- Capítulo 6: alberga los Smart Contracts utilizados a lo largo del proyecto que son necesarios para poder gestionar todos los requisitos planteados. En total se han desarrollado un total de siete contratos.

- Capítulo 7: se aborda el Front-End de la página Web que se ha desarrollado para la aplicación de *rent a car*. Se explican todos los componentes que se utilizan en el diseño de la aplicación, incluyéndose el esquema global jerárquico seguido para su desarrollo.

- Capítulo 8: contiene las pruebas de funcionamiento de la aplicación desarrollada para el proyecto. Se podrán ver todas las posibles interacciones entre las figuras de los clientes y de los administradores.

- Capítulo 9: En este último capítulo se reflexiona sobre las conclusiones que se han extraído a la hora de realizar este trabajo, así como las líneas de posibles mejoras a futuro.

- Apéndice A: En el apéndice se realiza la estimación económica del desarrollo del proyecto.

- Apéndice B: Se describe el proceso de instalación de todo el software necesario para el correcto funcionamiento de la aplicación.

- Apéndice C: En este último apéndice se reflejan todas las referencias bibliográficas utilizadas a lo largo del presente trabajo.

CAPÍTULO 2

Base teórica - Blockchain

2.1. Introducción.

En este capítulo se va a abordar los conceptos de Blockchain, su evolución desde sus orígenes hasta la actualidad, su funcionamiento, los distintos tipos de topologías y todos los componentes que interaccionan en estas redes. Por último, se expondrán los tipos de redes Blockchain que podemos encontrar actualmente.

2.2. ¿Qué es el Blockchain?.

Blockchain es un sistema de registro distribuido que promueve la descentralización, transparencia e integridad de los datos; es decir, es un sistema de almacenamiento de la información descentralizado en el cual cada nodo de la red tiene almacenado una copia exacta de cada modificación o interacción del sistema, y esto garantiza una disponibilidad de la información en cualquier momento, siendo imposible alterar esta, debido a que está replicada en todos los bloques de la que está compuesta la cadena de bloques (*Blockchain*) de la red. Estos bloques son espacios en los cuales se almacenan las transacciones que se generan en la red, y están pensados para optimizar el proceso de validación de las transacciones que se realizan. Además añadiendo la tecnología de los Smart Contracts, nos permite en las operaciones realizadas reducir significativamente el número de intermediarios.

De forma más técnica, Blockchain es un libro de registro en el cual se guardan y validan todas las transacciones en la cadena de bloques mediante los distintos algoritmos de consenso. Una vez añadido un bloque a la cadena de bloques, este no se puede cambiar. Los bloques se crean mediante funciones hash criptográficas que añaden seguridad. Si se desea cambiar el contenido de parte de la información que se encuentra en la cadena de bloques, debido a que esta es invariable, es necesario crear un nuevo bloque que se añadiría a la cadena existente, con la modificación de información realizada.

2.2.1. Ejemplos de utilización del Blockchain.

Hoy día hay grandes empresas apostando por esta tecnología para dar soluciones a distintos problemas que se les plantean. Se han seleccionado ejemplos que tienen algunas características similares a las que se van a utilizar en el presente proyecto. Son las siguientes:

Transacciones financieras.

- **Walmart** (Venta al por menor): utiliza la tecnología Blockchain para rastrear la gestión de productos de los agricultores en las tiendas.
- **Nestle** (Venta al por menor): uso de la tecnología Blockchain en la gestión de suministros en las tiendas de los productos de comida para bebés.
- **Tencent** (Comercio electrónico / Venta al por menor): solución para verificar la autenticidad de la factura y para garantizar el cumplimiento tributario.

Uso seguro de datos.

- **AIA Group** (Seguro): lanzó el primer tipo de banca seguros para compartir datos de pólizas.
- **British Airways** (Industria de viajes): implementan Blockchain para administrar los datos de vuelo y verificar la identidad de los viajeros.
- **Baidu** (Búsqueda): utiliza Blockchain para mejorar la gestión de los derechos intelectuales.

Trazabilidad de productos.

- **Alibaba** (Comercio electrónico): usan la tecnología Blockchain para rastrear productos de lujo en sus plataformas de comercio electrónico
- **Samsung** (Tecnología): utiliza la tecnología Blockchain para mejorar la gestión de la cadena de suministro cuando se trata de envíos de productos electrónicos.
- **BHP Billiton** (Minería): aprovechan la tecnología Blockchain para la gestión de la cadena de suministro.[48]

2.2.2. Ventajas e inconvenientes del Blockchain.

El uso de esta tecnología tiene una serie de ventajas e inconvenientes que desarrollaremos a continuación.

[48] «101blockchain,» [En línea]. Available: https://101blockchains.com/es/empresas-implementando-blockchain/. [Último acceso: 1 Julio 2021].

Ventajas:

- **Seguridad:** tiene un gran nivel de seguridad debido a que la información de los nodos se replica por toda la red y la hace prácticamente imposible de modificar. Únicamente sería posible si se modificara la información en todos los nodos que comprende la red Blockchain al mismo tiempo, evitando así que pudiesen comprobar la información y ver el fallo de seguridad, y eso es imposible.

- **Velocidad:** evita el uso de intermediarios a la hora de pagar o hacer transferencias de dinero, debido a que el propio sistema valida y autentifica las transacciones. De esta manera, se consigue que los pagos se puedan procesar en cualquier momento y de forma rápida. Esto permite que tenga una gran disponibilidad, debido a que está operativa los 365 días del año.

- **Transparencia:** las redes Blockchain son públicas y permite a todos los usuarios ver las transacciones y ver los bloques con total libertad, al contrario que los bancos tradicionales.

- **Inmutabilidad**: las transacciones que se guardan en los bloques que componen la red Blockchain no se pueden variar, lo que ofrece un grado de seguridad extra de la información que se almacena y hacia los usuarios que la utilizan.

Inconvenientes:

- **Costes elevados:** esto es debido a que los sistemas de aprobación de los bloques son automáticos, sin intermediarios y ahorrando tiempo y costes en el proceso. Se consigue utilizando un algoritmo de consenso (mecanismo por el cual la red alcanza el consenso). La mayoría de las redes Blockchain de gran relevancia (*Bitcoin*) utilizan el algoritmo de consenso Proof of Work (PoW), que presenta el inconveniente de un alto gasto energético y de tiempo.

- **Inalterable:** la cualidad de tener una red inalterable es una moneda de dos caras porque, debido a que las transacciones quedan reflejas y no se pueden revertir, si un usuario es estafado o robado, no podrá revertir la acción debido a la naturaleza de la red Blockchain.

- **Anonimato:** una característica de la tecnología Blockchain es que únicamente necesitas una Wallet o cartera para poder interaccionar con la red, pero debido al anonimato de las carteras, si se pierde la clave privada de las misma, será imposible recuperar la clave y el contenido de esta.

- **Tecnología en desarrollo:** debido a lo novedoso de esta tecnología, esto supone que aún no esté completamente desarrollada y se encuentre en fase de evolución, lo que repercute en que aún no se conozca todo su potencial. [49]

[49] «criptomoneda.ninja,» [En línea]. Available: https://criptomoneda.ninja/blockchain-ventajas-desventajas/#blockchain-ventajas-desventajas. [Último acceso: 1 Julio 2021].

2.3. Evolución del Blockchain.

Blockchain es una tecnología que cuenta con tan solo trece años de vida; es decir, es una tecnología muy joven y con un largo camino de desarrollo. En este corto período de tiempo ha evolucionado de forma considerable, encontrándose ahora mismo en la tercera etapa de su crecimiento. Estas etapas se pueden dividir en tres: del año 2008 al 2013, del 2013 al 2015 y del 2015 hasta 2021 (la actualidad).

- 1º Etapa Blockchain 1.0: esta etapa comprende desde 2008 hasta 2013, período dominado por Bitcoin. Fue creada por Satoshi Nakamoto que describió esta aplicación como un sistema electrónico peer-to-peer. Satoshi creó el bloque Génesis que fue el pilar utilizado para generar los demás bloques y unirlos a este, dando lugar a la cadena de bloques que conocemos actualmente.

- 2º Etapa Blockchain 2.0: esta etapa comprende desde 2013 hasta 2015. El causante del cambio de etapa fue Vitalik Buterin, uno de los contribuyentes del desarrollo de la Blockchain Bitcoin. Creó Ethereum como una red Blockchain maleable que puede realizar más acciones que ser únicamente una red peer-to-peer. La creación de Ethereum fue muy importante, debido a que con ella vinieron los denominados Smart Contracts, que son códigos que se despliegan en la red y que son ejecutados de forma automática sin necesidad de intervención humana de ningún tipo. Cuando decimos "desplegar", nos referimos a que son enviados a la red Blockchain para que se almacenen y se ejecuten de manera indefinida cuando se den las condiciones necesarias.

  El uso de los Smart Contracts en la red Ethereum hizo que se convirtiese en la Blockchain que procesa un mayor número de interacciones diarias.

- 3º Etapa Blockchain 3.0: esta etapa transcurre desde 2015 hasta 2021 (la actualidad). Todavía es un poco incierto cuál será la innovación que consiga destacar, como lo hicieron en su momento Bitcoin y Ethereum en sus respectivas etapas. Se están creando numerosas redes Blockchain con funcionalidades dispares, aunque se están caracterizando por dejar de usar el algoritmo de consenso Proof of Work (PoW) que tiene un elevado gasto energético y medioambiental, y se está optando por pasar a otros algoritmos de consenso como Proof of Stake o Delegated Proof of Stake.

  Ethereum actualmente está en proceso de pasar de usar el algoritmo de consenso Proof of Work al algoritmo de consenso Proof of Stake (PoS) en un hard fork o bifurcación que pretenden realizar a lo largo de este año (2021). Eso conllevaría al segundo hard fork que han realizado desde su nacimiento, siendo el primero el de la separación de Ethereum con Ethereum Classic (esto se desarrollará más adelante).

  En cuanto al algoritmo de consenso Delegated Proof of Stake (DPoS), hay numerosas Blockchain que están optando por su implementación por su ahorro energético y bajos costes para los usuarios. También tienen en común su alta escalabilidad, que es algo

que la red Bitcoin tiene grandes dificultades para suplir. Algunas de estas redes son: EOS, Lisk, Ark y Tron, que se crearon con fines muy diversos. [50]

2.4. Funcionamiento de Blockchain.

Blockchain (cadena de bloques) a una tecnología que nos permite crear un registro o libro de contabilidad distribuido en una red descentralizada, sin la necesidad de una autoridad reguladora y central, que mantiene la garantía de seguridad e integridad de la red.

Se basa en generar transacciones para luego poder asentarlas dentro de un bloque. Cuando un bloque se completa, aparece como actor en la transacción el minero, figura que valida la información guardándola definitivamente en el bloque a cambio de una recompensa. La operación de minar sirve para poder validar los bloques y así poder introducirlos en la cadena de bloques.

Los bloques se pueden crear a la hora de hacer transacciones. Una transacción puede ser una transferencia de tokens (dinero) de una cuenta a otra cuenta, como sería el caso de Bitcoin. Otro tipo de transacción puede ser el cambio de una variable de un programa, que se encuentra en la red desplegada previamente, o un comentario. Cada modificación que se haga se considera una transacción, y cuando haya un conjunto de transacciones se creará un bloque con estas para poder minarlo posteriormente.

A continuación, vamos a ver otro ejemplo de forma más gráfica.

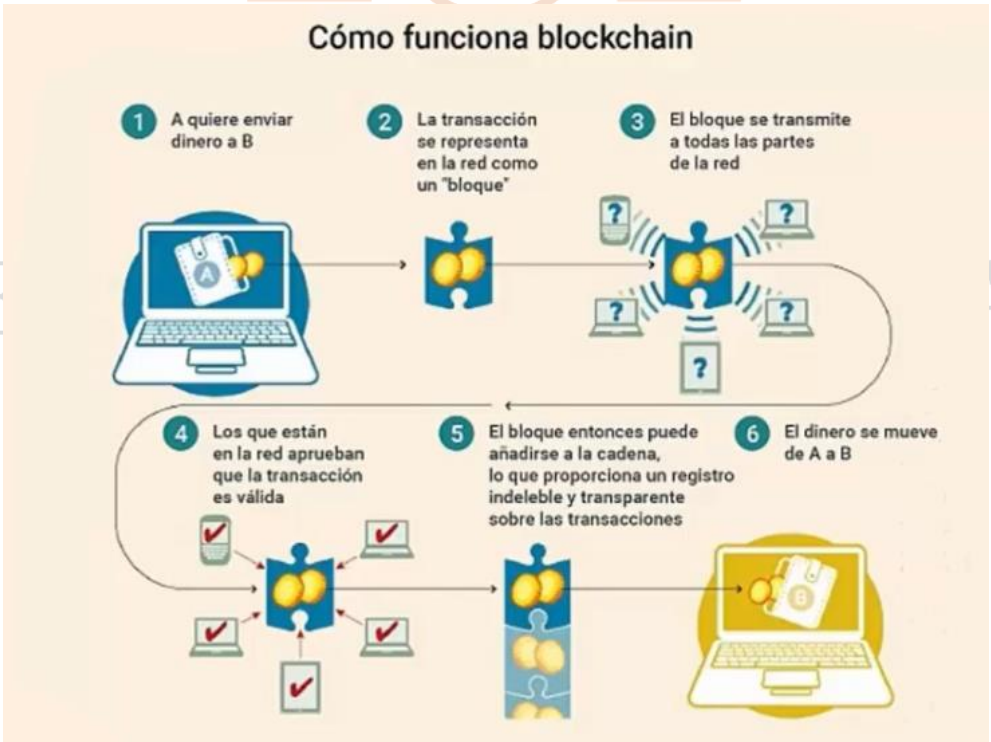

Figura 2: Funcionamiento de una transacción [28]

[50] «101blockchains,» [En línea]. Available: https://101blockchains.com/es/historia-de-la-blockchain/. [Último acceso: 1 Julio 2021].

El ejemplo anterior se puede dividir en seis pasos:

1. La persona "A" quiere hacer una transferencia de tokens a "B".
2. La transacción, que ha hecho "A", se guarda en un bloque.
3. El bloque recién creado se distribuye por toda la red.
4. Los mineros actúan minando el bloque y, por ende, verificando la validez de este.
5. Una vez minado el bloque este se introduce al final de la cadena de bloques y queda fijado para siempre.
6. "B" recibe los tokens que le ha mandado "A".

Aunque parezca complicado en un principio, gracias a todos los pasos que se han realizado, se puede garantizar la seguridad y la integridad de la transacción.

2.5. Tipos de arquitectura de redes.

Las arquitecturas sirven para estructurar nuestras redes. Dependiendo del modelo, podemos englobarlo en uno de los tres tipos que existen:

- La red centralizada: consiste en un nodo central que cuenta con canales en los que están conectados el resto de los nodos. Cuando un nodo quiere interactuar con otro nodo lo tendrá que hacer a través del nodo central y sus canales. Si el nodo central sufre algún tipo de daño, y queda inoperativo, la red caería debido a que es de vital importancia para el funcionamiento de la misma. Este modelo es el que ha sido usado más extensamente, aunque recientemente está siendo sustituida por el modelo distribuido y el descentralizado.

- La red descentralizada: se caracteriza por tener más de un nodo central. Existe un centro colectivo con diversos puertos de conexión y presenta una topología de árbol. Eso quiere decir que, si uno de los nodos reguladores cae o se desconecta, no conlleva que el resto de los nodos de la red pierdan conectividad y, por ende, la red no cae ante la pérdida de nodos.

- La red distribuida: tiene como característica principal la ausencia de un centro individual o colectivo. Los nodos se unen entre ellos de forma que ninguno tiene un mayor peso que otro. Cada nodo es independiente y puede moverse libremente. Esto conlleva que los nodos no tengan el poder de filtrar la información que se transmite por la red. [1] [2]

[28] «SECURED COIN,» [En línea]. Available: https://invertirencriptomonedas.es/que-es-una-criptomoneda [Último acceso: 17 Febrero 2021].
[1] «Desarrollo activo,» [En línea]. Available: https://desarrolloactivo.com/blog/centralized-decentralized-distributed-p2p/. [Último acceso: 17 Febrero 2021].
[2] «Desde linux,» [En línea]. Available: https://blog.desdelinux.net/descentralizar-internet-redes-descentralizadas-servidores-autonomos/. [Último acceso: 17 Febrero 2021].

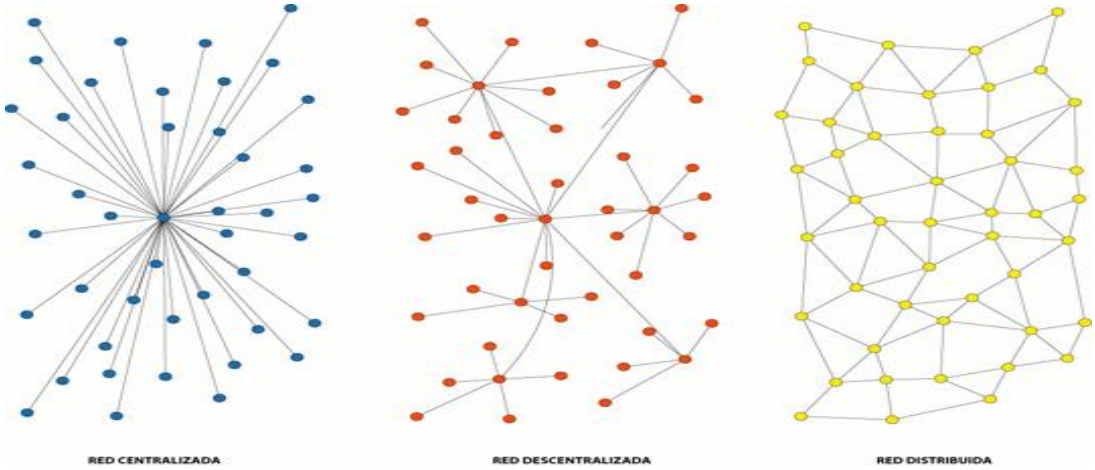

Figura 1: Topologías de redes [27]

2.6. Bloques.

Un bloque es un conjunto de transacciones realizadas por los usuarios que se encuentran en interacción con la red Blockchain. Se compone de tres partes claramente diferenciadas:

- Un código alfanumérico que lo enlaza con el bloque anterior. Este es el *hash* del bloque previo para poder identificarse como el siguiente.
- El "paquete" de transacciones que incluye (cuyo número viene determinado por diferentes factores)
- Otro código alfanumérico que enlazará con el siguiente bloque, y que es el hash que el minero tendrá que obtener.
- Un número de una longitud extensa que representa la fecha y la hora completa en la que se minó el bloque. Este número es el tiempo en segundos, desde 1 de enero de 1970. A este dato se le denomina *Timestamp*.
- *Nonce*, es un número aleatorio que se utiliza en el protocolo de autenticación. Se trata de una combinación de números con el hash para evitar la manipulación de la información del bloque.

En este ejemplo que se muestra a continuación, se puede ver cómo se relacionan los bloques, y cómo un bloque puede contener varias transacciones en su interior.

[27]«DigiByte HISPANO,» [En línea]. Available: https://www.digibytehispano.org/como-funciona-una-red-descentralizada/. [Último acceso: 18 Febrero 2021].

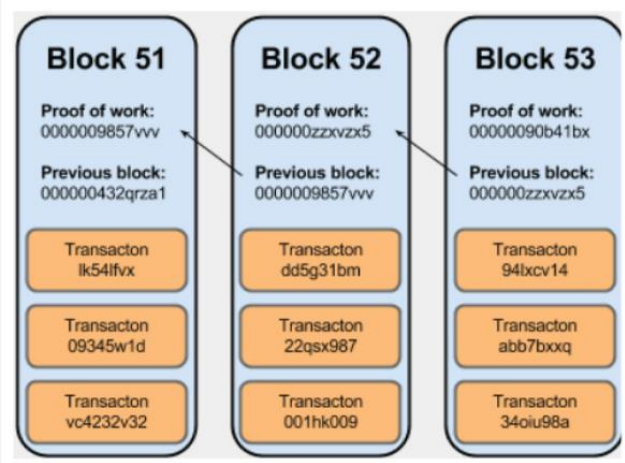

Figura 3: Relación entre bloques [47]

Explicada la estructura que necesitan tener los bloques una vez minados para poder introducirse en la cadena de bloques, procedemos a exponer una breve explicación de un bloque en particular: el *bloque Génesis*.

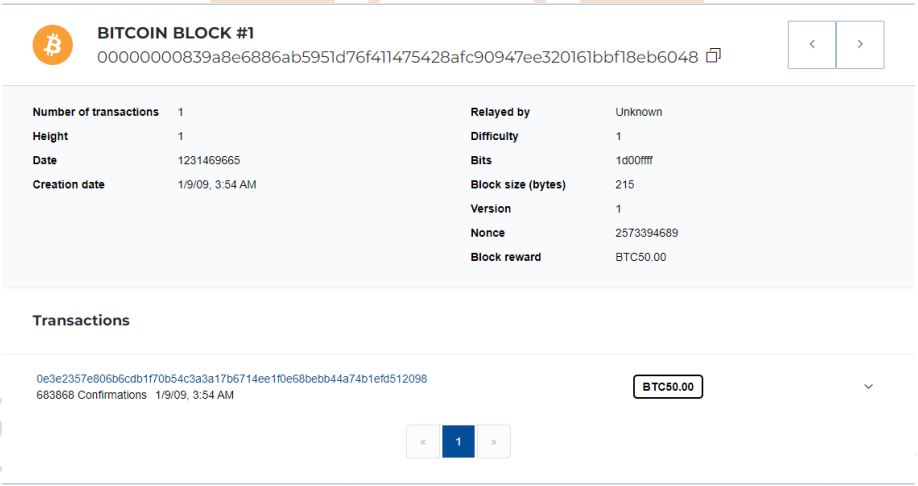

Figura 4: Bloque Génesis de Bitcoin [30]

Este bloque es uno especial, debido a que es el *bloque Génesis* de la red Bitcoin. También se le puede denominar por otro nombre, como el *bloque cero*. Fue creado por Satoshi Nakamoto en el año 2009. Al primer bloque de una red Blockchain se le llama *bloque Génesis*, debido a que es el bloque que da pie a la generación de la cadena de bloques.

[47] F. Canós, «dairioabierto,» [En línea]. Available: https://www.diarioabierto.es/378175/los-bloques-del-blockchain [Último acceso: 22 Marzo 2021].

[3] «Ethereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/. [Último acceso: 22 Marzo 2021].

[30] «bit2me,» [En línea]. Available: https://explorer.bit2me.com/btc/block/00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048. [Último acceso: 22 Marzo 2021].

2.7. Mineros.

Los mineros son computadores o equipos electrónicos especializados cuya función es aportar poder computacional o de minar a la red Blockchain. La necesidad de ese poder computacional o de minado es permitir validar las transacciones que se llevan a cabo por las entidades y así poder constituir un bloque. La misión del minero es calcular la función hash criptográfica que corresponde al conjunto de transacciones que engloba el bloque. Una vez completado se le da como recompensa, una cantidad de tokens de la moneda minada.

2.8. Nodos.

Un nodo es un ordenador conectado a la red Blockchain, que dispone de un software con capacidad de almacenar y distribuir una copia en tiempo real de la cadena de bloques actualizada. Este verifica todas las transacciones que hay en cada bloque manteniendo así la integridad de la red. Existen tres tipos de nodos: *nodos completos, nodos de ligero* y *los nodos de archivo.*

Los *nodos completos* son nodos que almacenan todos los datos de la Blockchain, participa en la validación y verifica los bloques. También sirve a la red y proporciona los datos que le soliciten.

Los *nodos de ligero* almacenan únicamente la cadena de encabezados y el resto lo solicita. Está pensado para dispositivos de capacidad limitada como *smart phones* o dispositivos integrados.

Los *nodos de archivo* actúan, igual que los nodos completos, almacenando toda la información de la red Blockchain, pero además crean un archivo de datos históricos que permite hacer consultas más concretas, como los saldos de las cuentas en un bloque en concreto. [3]

2.9. Funciones hash.

En las ciencias de la computación, "hashing" tiene muchos significados. En cambio, dentro de la criptografía, una función hash *h(x)* es una función que se utiliza para poder comprobar la integridad de los datos presentes en los bloques, y se ubican al principio del bloque y al final de este. El hash, que se encuentra al principio del bloque, es el hash del bloque anterior y se coloca ahí para poder relacionarlo con el bloque previo. El otro hash, que se sitúa al final del bloque, corresponde con el hash del bloque completo. Esto se hace así para poder garantizar la seguridad y la integridad del bloque. Para que se considere un hash, *h(x)* tiene que cumplir los siguientes aspectos:

- **Compresión:** Dado una longitud de entrada cualquiera (*x*), la salida de *y = h(x)* debe tener una longitud menor. Esta longitud suele ser fija a 160 bits, independientemente de la longitud de entrada.

- **Eficiencia:** Debe ser "sencillo" calcular *h(x)* dado *x*. Este grado de complicación aumentará con la longitud de x, pero esta no aumentará de forma esporádica.

- **Unidireccional o una sola dirección:** Dado el valor de *y* no es posible computacionalmente encontrar un valor de *x* tal que *h(x) = y*; es decir, no es posible invertir la función hash.

- **Resistencia débil ante colisiones**: Dado el valor *x* y el valor *h(x)* es imposible encontrar un valor *y*, tal que $y \neq x$ y se consiga *h(y) = h(x);* es decir, no se puede modificar el mensaje sin modificar el valor del hash calculado.

- **Resistencia fuerte ante colisiones:** Es imposible encontrar un *y* un *x* que cumplan $y \neq x$ y que cumplan a su vez *h(x) = h(y); e*s decir, no puede haber dos valores de entrada distintos que den el mismo valor de salida. [4]

En la siguiente imagen, se puede apreciar cómo a cada entrada la función criptográfica hash produce un valor distinto. También se puede comprobar cómo al cambiar mínimamente la entrada del rectángulo dos, que tiene el mensaje "*The red fox jumps over the blue dog*", y el rectángulo tres, con el mensaje "*The red fox jumps ouer the blue dog*", produce un valor completamente distinto.

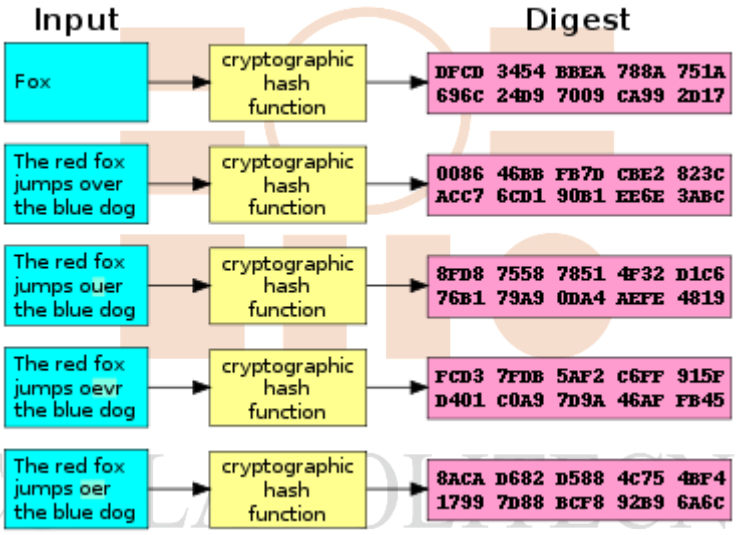

Figura 5: Ejemplo función criptográfica hash [29]

Observando la imagen anterior, se puede apreciar que se cumplen todas las propiedades mencionadas anteriormente, como la compresión. Se puede percibir cómo independientemente de la longitud de entrada *o input*, la salida *o digest* es siempre la misma. Otra propiedad que se puede ver es su resistencia débil ante colisiones, donde al variar mínimamente la entrada conseguimos una salida completamente distinta.

[4] M. STAMP, Information security: principles and practice, John Wiley & Sons, 2011.

[29] «GENBETA,» [En línea]. Available: https://www.genbeta.com/desarrollo/que-son-y-para-que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales. [Último acceso: 22 Febrero 2021]

2.10. Árbol de Merkle.

Es necesario introducir el concepto del árbol de Merkle porque es la estructura que van a tener las denominadas "cadenas de bloques", a las que hemos hecho referencia.

Un árbol de Merkle o árbol hash es una estructura de datos en árbol que tiene como finalidad relacionar cada nodo con una raíz única asociada a este. Para poder conseguirlo, cada nodo, que no es una hoja del árbol, está etiquetado con el hash de sus nodos hijos. A los nodos hijos se les denomina con el nombre de *hoja*, y a los nodos padres con el nombre de *rama.* Esta estructura de nodos hijos con nodos padres se repite de forma continua hasta llegar al nodo raíz o root Merkle.

Este diseño fue creado por Ralph Merkle**,** en el año 1979, para poder agilizar los procesos de verificación de grandes cantidades de información.

Los árboles de Merkle tienen las siguientes propiedades:

- **Eficiencia en la verificación de datos:** esto es debido a su estructura en árbol, que permite relacionar un único punto (el nodo raíz) con varios datos superiores. Gracias a esto, si se verifica la validez del punto, se puede asegurar la validez de la estructura del árbol entero.

- **Adaptabilidad:** este tiene una enorme adaptabilidad y permite utilizarlos en muchos contextos distintos. Estos pueden ser en bases de datos, en redes distribuidas (*Peer to Peer*), sistemas de archivos, estructuras de llaves públicas, entre otros ejemplos.

- **Eficiencia sincronización de datos:** esto es debido a que se pueden sincronizar los datos a través de diferentes nodos en un sistema distribuido. Este método es eficiente porque, para comparar los datos y poder observar cuales han sido modificados o actualizados, no es necesario ver los datos como tal, sino ver los valores hash para descubrir si han sido modificados. Si este fuera el caso, se envían a través de la red y se actualizan en el resto de los nodos, consiguiendo así una sincronización perfecta.

A continuación, se muestra una figura con una representación gráfica de un árbol Merkle.

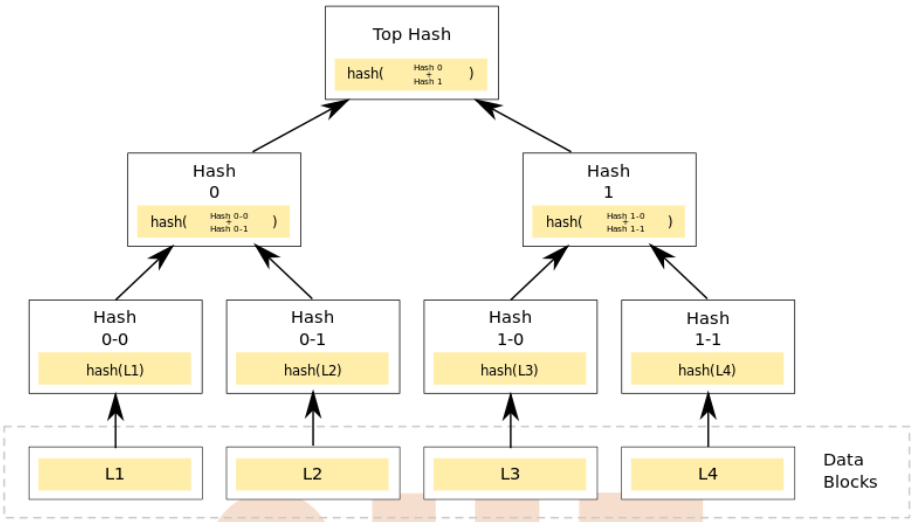

Figura 6: Estructura de un árbol de Merkle [31]

Se puede observar cómo las "hojas" están relacionadas con los nodos superiores llevando la información de estos para poder después verificar, si fuese necesario, la procedencia de los nodos hijos.

En la actualidad, su uso está enfocado en redes Blockchain, debido a la naturaleza de esta. Un cliente de la red puede descargarse el historial completo de la Blockchain, o bien puede descargarse solo una parte de esta para reducir el tamaño de la descarga. Descargarse solo una parte de la Blockchain no le resta seguridad al cliente, debido a que el cliente puede descargar en concreto un nodo raíz, y como este nodo esta enlazado con otros bloques anteriores a él, solo tiene que verificar esos últimos.

2.11. Algoritmos de consenso.

Un algoritmo de consenso es el mecanismo por el cual la red alcanza el consenso. Estos mecanismos son procesos de toma de decisiones, que son necesarios debido a la naturaleza descentralizada de la red Blockchain. Se utilizan para la creación de bloques y para su posterior adhesión a la cadena de bloques.

El procesamiento de estos algoritmos se realiza en equipos de minado, utilizando para ello procesamiento por GPU´s. Debido al aumento de la dificultad a la hora de minar, la capacidad de procesado de las CPU´s (pensadas para la ejecución del código en serie) las convierte en el hardware menos adecuado. Sin embargo, las GPU´s tienen la capacidad de calcular miles de datos en paralelo, por lo que no solo pueden estar minando diferentes partes de la cadena de

[31] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle#/media/Archivo:Hash_Tree.svg. [Último acceso: 27 Marzo 2021].

bloques de cada criptomoneda sino que, además, pueden aplicar diferentes algoritmos y partes de manera colaborativa. [5]

El algoritmo de consenso usado en cada momento dependerá del tipo de red Blockchain utilizada. Estos son ejecutados por los actores involucrados en el consenso, que pueden ser la figura de *delegado* o *minero*. Hay muchos tipos de algoritmos de consenso, pero nos centraremos en los más extendidos a nivel de redes Blockchain.

2.11.1. Proof-of-Work (PoW).

Proof of Work o prueba de trabajo. Es el primer algoritmo de consenso y por tanto el más antiguo. Se emplea en distintas redes como el Bitcoin o Ethereum.

Se basa en que los mineros utilizan sus equipos de minado para resolver unos problemas matemáticos complejos Estos problemas matemáticos se denominan acertijos criptográficos. El minero con mayor capacidad de minar (*hashrate*) tendrá una mayor posibilidad de resolver el problema en primer lugar. Para poder resolver el acertijo criptográfico, cada minero calcula el resultado de aplicar la función de hash a la cabecera del bloque. Como se ha mencionado antes, la cabecera del bloque tiene un campo que se denomina *nonce*. Los mineros varían el valor de este campo para obtener distintos valores de hash, hasta conseguir que el resultado comience con tantos ceros como estén establecidos.

El minero que resuelva primero el problema, expandirá a través de la red la solución para que el resto de los nodos puedan copiar esa información en su cadena de bloques y, como compensación, este recibirá una cantidad significativa de la criptomoneda que estuviera minando.

Este modelo presenta una gran desventaja, y es su gran consumo de energía debido a la potencia de cómputo requerida para resolver el acertijo criptográfico.

Este modelo de consenso es vulnerable al ataque del 51% o ataque mayoritario. Este ataque consiste en que un minero tenga, al menos, un 51% del cómputo de la red Blockchain, lo que posibilitaría que pudiese modificar los bloques que calcula. Esto le permitiría ser la entidad que los calcularía y validaría, y eso le permitiría realizar diversos tipos de ataques. Debido a esta vulnerabilidad, se creó el modelo PoS.

2.11.2. Proof-of-Stake(PoS).

Proof of Stake o Prueba de Participación. Se creó para evitar el problema que tenía el PoW, que es su vulnerabilidad al ataque del 51%.

PoS tiene un único requisito para poder ser validador, y es que los usuarios apuesten su Eth (si estamos en la red Ethereum) para poder convertirse en validadores de la red. La figura del

[5] «Hard Zone,» [En línea]. Available: https://hardzone.es/noticias/tarjetas-graficas/mineria-via-gpu-motivos/. [Último acceso: 27 Marzo 2021].

validador es la misma que la del minero, con las mismas funciones y posibles recompensas. La única diferencia, como se ha mencionado antes, es que previamente tienen que aportar parte de su patrimonio en criptodivisas (ether) para poder desempeñar el trabajo.

Paso a explicar las diferencias entre ambos modos de forma más clara. Un ejemplo podría ser que invirtiésemos 1000 euros en la adquisición de equipos de minado para el modelo PoW, mientras que en el modelo PoS, la inversión de los 1000 euros se realizaría en adquirir criptomonedas, y usar éstas como depósito para poder comprar una cantidad equivalente de creación de bloques.
Las ventajas que presenta este modelo son las siguientes:

- Mayor eficiencia energética: en este modelo no es necesario un gasto energético tan elevado como en el caso de PoW.
- Bajos requisitos Hardware: no es necesario tener computadores especializados de última generación para poder minar y crear nuevos bloques.
- Mayor descentralización: la necesidad de prueba de participación debería producir un mayor número de nodos en la red.
- Mayor soporte para cadenas de fragmentos: es una mejora clave para aumentar la red de Blockchain.

El peligro de recibir un ataque del 51% aún es posible, pero es muy arriesgado para los atacantes. Esto es debido a que, para efectuarlo, es necesario, por parte del atacante, controlar el 51% de Eth (si estamos apostados en la red Etherium). Aparte de que es un capital monetario importante, seguramente haría que el valor de la moneda se devaluase y perdiera parte de su inversión. Aunque el problema real es que existe la posibilidad de baneo o expulsión de la red, perdiendo todo lo invertido hasta el momento.

Se puede resumir en una frase dicha por Vlad Zamfir: “Esto es como un minero que participa en un ataque del 51%, lo que hace que su hardware de minería se queme de inmediato” [6]

2.11.3. Delegated Proof-of-Stake(DPoS).

Delegated Proof of Stake o Prueba de Participación Delegada. Es considerada como una versión más eficiente y democrática que Prueba de Participación (PoS). En este modelo aparece la figura del *delegado*, que son las entidades que tienen el poder de crear bloques para añadirlos a la Blockchain.

Este modelo cuenta con un sistema de votación en tiempo real, donde todos los integrantes de la red eligen por votación a unos delegados. Estos delegados tienen un poder de voto proporcional a su participación en la misma; es decir, cuanto mayor sea el número de créditos que posea mayor será el número de votos.

Los delegados definen una rotación de líderes. Esto quiere decir que cada delegado tiene un puesto en la rotación para permitirle crear un bloque, y una vez creado pasa al siguiente delegado. Si un delegado en su turno no crea ningún bloque, después de un tiempo, pasa al siguiente delegado, y así sucesivamente de manera cíclica.

[6] «Ethereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#:~:text=Pros%20and%20cons%20%20%20%20Pros%20,secure%20sharding.%20Shar%20...%20%20%20%20%20. [Último acceso: 27 Marzo 2021].

Este sistema resuelve un problema clave de las redes Blockchain: la escalabilidad. Hay que recordar que este modelo no necesita equipos de altas prestaciones ni un consumo de energía elevado como en otros modelos. Esto presenta una serie de ventajas y desventajas que veremos a continuación.

Ventajas:

- Descentraliza la participación: descentraliza la participación en la red, debido a que éste relaciona la participación en la red con el número de créditos o tokens que tiene en esta cada usuario.
- Eficiencia: el sistema de elección de delegados en cada ronda se realiza de forma rápida, lo que permite una escalabilidad notable de la red.
- Optimización
- Seguridad en tiempo real: cualquier intento de atentar contra la red, o de manipulación de alguna parte de esta, se detecta de inmediato por los votantes y expulsa al delegado que lo estuviera causando.

Desventajas:

- Coordinación necesaria: un funcionamiento correcto de la red requiere una coordinación de la comunidad para un gobierno eficaz. Eso quiere decir que, por otro lado, es más sencillo preparar un ataque. Esto es debido a que, como el número de delegados es muy inferior al número de participantes, es mucho más sencillo originar un ataque del 51% pudiendo así modificar la mayoría de los bloques que se creen; todo claro está, si los votantes no se dan cuenta de las intenciones maliciosas de estos delegados.
- Posibilidad de ser un sistema centralizado: aunque en principio está concebido como un sistema descentralizado, el número bajo de delegados y la posibilidad de ausencia de votante pueden convertirlo en un sistema centralizado.

Estos son los modelos de consenso más extendidos en la actualidad, aunque existen muchos, como el *Leased Proof of Stake (LPoS)* o el *Practical Byzantine Fault Tolerance (PBFT),* entre otros.

Por último, vamos a incluir una pequeña descripción gráfica del resto de los algoritmos de consenso. Aunque no se vayan a utilizar en este proyecto, siempre es interesante conocerlos para así tener una mejor idea de la importancia de los descritos previamente.

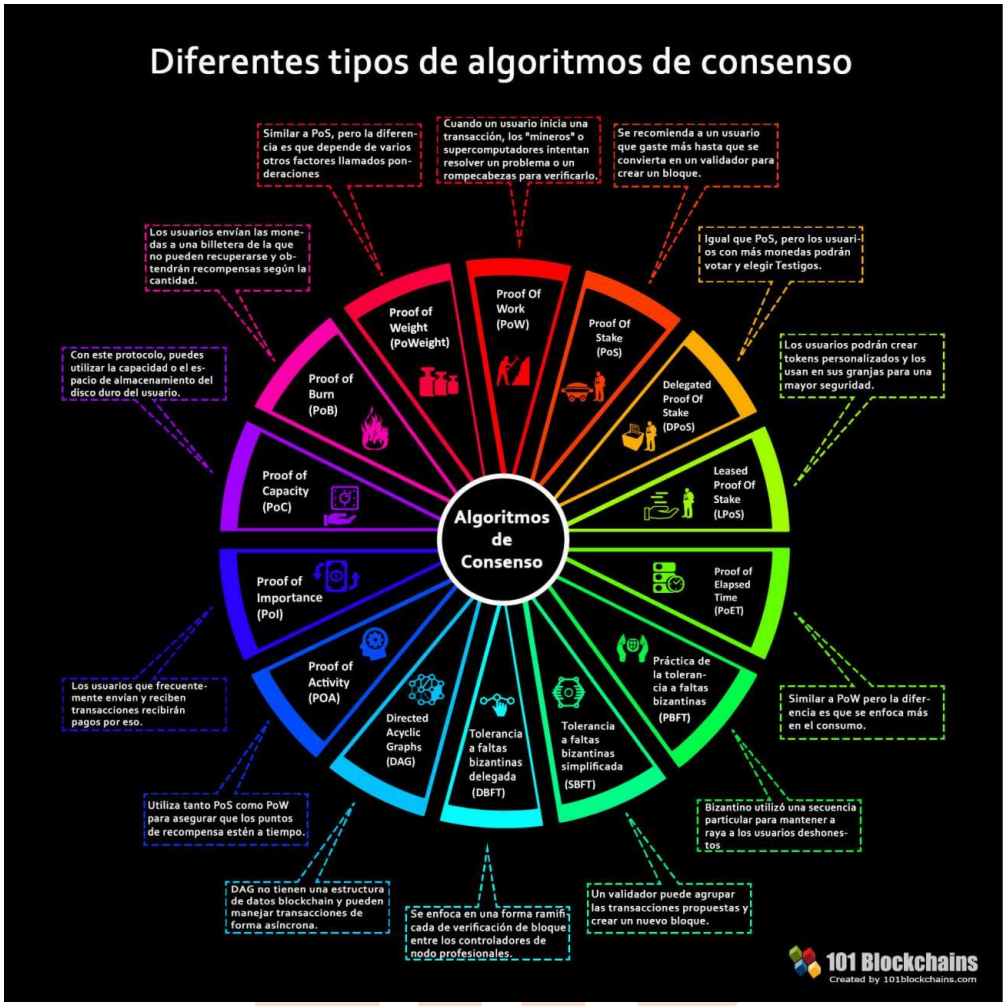

Figura 7: Algoritmos de consenso [32]

El resto de los algoritmos de consenso, que están estandarizados y se consideran importantes, son los siguientes:

- Leased Proof Of Stake:
- Proof of Elapsed Time:
- Practical Byzantine Fault Tolerance (PBFT)
- Tolerancia Bizantina a Fallos Simplificadas (SBFT)
- Tolerancia Bizantina a Fallos Delegada (DBFT)
- Directed Acryclic Graphs (DAG)
- Proof of Activity (POA)
- Proof of Importance (POI)
- Proof of Capacity (PoC)
- Proof of Burn (PoB)
- Proof of Weight (PoWeight)

[32] N. Rodriguez, «101 Blockchains,» [En línea]. Available: https://101blockchains.com/es/algoritmos-de-consenso-blockchain/#1. [Último acceso: 1 Abril 2021].

2.12. Tipos de redes.

Podemos distinguir tres tipos de redes Blockchain: *las redes públicas, las redes privadas y las redes de consorcio o permisionadas*. Todas sirven para poder trabajar con distintas modalidades de Blockchain, dependiendo de las necesidades del cliente en ese momento. A continuación, pasamos a describir los tres tipos que existen:

2.12.1. Redes Públicas.

Las redes públicas son redes con las siguientes características:

- Cualquier persona puede descargarse el código y administrar un nodo público en su máquina local validando transacciones en la red y participando en el proceso de consenso, lo que hace que cualquiera tenga derecho a participar en el proceso de los bloques que se introducen en la cadena.

- Cualquier persona puede realizar transacciones en la cadena. Cualquier transacción válida será añadida.

- Cualquier persona puede acceder y ver las transacciones utilizando un explorador de bloques; sin embargo, estas transacciones, aunque son públicas, son anónimas. [7]

Este tipo de redes Blockchain utilizan el algoritmo de consenso de *Proof Of Work (PoW)* en el que, al ser una red pública, cualquiera puede participar en él.

Este tipo de redes presentan una serie de ventajas. Cualquier persona puede involucrarse en la red y utilizarla para sus fines particulares, por ejemplo utilizando DApps (Aplicaciones Descentralizadas).

Las Blockchain, de este tipo más conocidas, son las siguientes:

a) Bitcoin:

Bitcoin es un sistema creado por Satoshi Nakamoto en 2008. Fue la primera aplicación desarrollada sobre Blockchain y fue diseñada como un sistema de pago descentralizado, seguro y rápido. Posibilita el intercambio de criptomonedas. En este caso el token utilizado es el bitcoin, un tipo de moneda virtual que funciona como moneda global y accesible por todo el mundo y que compite contra el dinero *fiat* o el oro. Sin embargo, Bitcoin tiene una cantidad finita de monedas con una cantidad de 21 millones, lo que podría hacer que a priori sea deflacionaria y a largo plazo pierda su valor (aunque actualmente, la especulación está alterando significativamente su valor).

b) Ethereum:

Esta plataforma fue fundada por Garvin Wood, Jeffrey Wilcke y Vitalik Buterin. Es una de las plataformas descentralizadas más consagradas de Blockchain en código abierto disponibles en la actualidad. Es famosa por su robusta funcionalidad y flexibilidad a la

[7] «ABANCA innova,» [En línea]. Available: http://abancainnova.com/es/opinion/los-tipos-de-blockchain-publica-privada-o-consorcio-explicados/. [Último acceso: 2 Abril 2021].

hora de crear contratos inteligentes. Se usa ampliamente en múltiples casos de uso de la industria y tiene su propia moneda (ether).

c) Hyperledger:

Hyperledger es una plataforma de código abierto creado por la Fundación Linux. Tiene el objetivo de crear un ecosistema centrado en obtener soluciones de código abierto en el ámbito empresarial, para conseguir así estándares de desarrollo y expansión.

d) Corda:

Corda se ha desarrollado para atender las necesidades específicas de los servicios financieros, pues fue desarrollada por un consorcio de más de 70 identidades financieras de todo el mundo con código abierto y espera ser el estándar de este sector para la tecnología Blockchain.

e) Ripple:

Ripple fue creada como una red de pagos digitales para transacciones financieras al instante. Desde su creación, se ha especializado en pagos entre distintos países. Funciona como una base de datos distribuida que permite ejecutar transacciones entre dos o más partes con las características que ofrece Blockchain de verificabilidad, transparencia y permanencia.

2.12.2. Redes Privadas.

Las redes privadas son redes que carecen de las características antes mencionadas de la red pública. En estas redes, para poder escribir en la Blockchain, debes pasar una serie de filtros o pertenecer al órgano que se encarga de escribir en la cadena de bloques. La lectura puede ser abierta para todo el mundo, o por el contrario también puede tener algunas restricciones de acceso.

Las redes privadas se utilizan principalmente a nivel de empresa, siendo muy complicado que se extienda a un nivel superior. Las aplicaciones que suelen estar en este tipo de redes son administraciones de bases de datos o auditorias. Al ser una red privada, el algoritmo de consenso que se utilice no es necesario que sea el *Proof of Work (PoW)* como en las redes públicas. La ventaja principal sería su simplificación en la cadena de bloques y la administración de los datos, debido a su reducida escala.

2.12.3. Redes de consorcio o permisionadas.

Las redes de consorcio están en un punto medio entre las redes públicas y las redes privadas. Se puede describir como una red privada con parte de pública. El “poder” no se ubica en una sola entidad, sino que está repartida en un grupo de personas o entidades.

Las redes de consorcio se diferencian de las redes públicas en que, las primeras, no permiten que todo el mundo participe en el intento de verificar las transacciones (minado). Esto permite que al evitar eso, éstas sean más rápidas y tengan una mayor escalabilidad y privacidad de las transacciones.

Suelen utilizar el mecanismo de consenso de *Proof of Stake (PoS),* debido a que las entidades verificadoras son una serie de nodos que han sido elegidos previamente, y cuentan con la confianza de la red entera.

Están enfocadas principalmente para conglomerados de empresas, porque para poder verificar una transacción debe ser votada a favorablemente por la mayoría de los integrantes de la red.

Una de las Blockchain permisionadas más popular es la siguiente:

a) Alastria:

Alastria es una red permisionada, promovida por empresas e instituciones sin ánimo de lucro. Es una plataforma colaborativa de ámbito nacional, disponible para emprendedores, grandes empresas, instituciones públicas y Pymes entre otros.

El objetivo de Alastria es la aceleración de los ecosistemas digitales a nivel nacional que contribuyan a un desarrollo prometedor de nuestra industria y tecnología. [8]

[8] «Avance Digital,» [En línea]. Available: https://avancedigital.mineco.gob.es/es-ES/Participacion/RespuestasEstatregiaDigital/asociacion-alastria.pdf#:~:text=Alastria%20es%20un%20consorcio%20multisectorial%20sin%20%C3%A1nimo%20de,de%20una%20red%20semip%C3%BAblica%20permisionada%20Blockchai%20%28. [Último acceso: 2 Abril 2021].

CAPÍTULO 3

Base teórica - Ethereum

Tras haber desarrollado los conceptos básicos de las redes Blockchain y los distintos tipos que existen, nos vamos a centrar en la red Ethereum. A diferencia de otras Blockchains, esta red nos permite crear y desplegar contratos para luego poder utilizarlos libremente. El concepto de qué es "un contrato" se desarrollará más adelante. [26]

Figura 8: Icono de Ethereum [33]

3.1. Definición.

Ethereum es una red Blockchain de código abierto que se utiliza para desplegar contratos inteligentes. Estos son programas escritos con un lenguaje de programación específico que se despliegan en la red y que siempre están activos, lo que permite una gran flexibilidad y adaptabilidad. Fue desarrollado por Garvin Wood, Jeffrey Wilcke y Vitalik Buterin.

[26] «Ethrereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/. [Último acceso: 24 Abril 2021].
[33] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Ethereum#/media/Archivo:ETHEREUM-YOUTUBE-PROFILE-PIC.png. [Último acceso: 8 Abril 2021].

El token es la moneda que se utiliza para trabajar en las redes Blockchain. Es el sustitutivo del dinero tradicional. En un principio, la moneda utilizada por la red Ethereum es el ether, pero gracias a su gran flexibilidad y adaptabilidad, la red Ethereum nos permite crear nuestra propia moneda o token y usarlo de igual manera que el ether, siempre y cuando todas las partes implicadas acepten el uso de la moneda o token.

Funciona de manera descentralizada a través de una máquina virtual llamada Ethereum Virtual Machine (EVM). Esta máquina virtual nos permite ejecutar los contratos creados con el lenguaje de alto nivel, llamado Solidity.

El funcionamiento de Ethereum es muy parecido al de Bitcoin a través del Blockchain, pero con algunas diferencias. Ethereum incluye un nuevo concepto que lo hace diferente al resto: los contratos inteligentes o *Smart Contracts*. Se desarrollarán más adelante, pero para entenderlo por ahora, se podría decir que es un contrato entre dos partes que siempre está vigente, y está esperando a que se cumplan ciertos requisitos para poder ejecutarse de forma automática. [9]

3.2. Ethereum vs Bitcoin.

Aunque ambas sean redes Blockchain de tipo pública, se distinguen en muchos aspectos. Para empezar, las separan unos años de diferencia puesto que Bitcoin fue creada en 2008, mientras que Ethereum nació en 2013. Bitcoin nació como una plataforma de pago descentralizado, mientras que Ethereum buscaba ser una plataforma donde ejecutar código y contratos inteligentes. Al ser más moderna, Ethereum usa un algoritmo criptográfico llamado Ethash, mientras que Bitcoin utiliza el algoritmo criptográfico Sha256.

Cada plataforma tiene su propio token. Bitcoin utiliza un token llamado Bitcoin y Ethereum utiliza Ether. Cada moneda tiene sus propias divisiones que se podrán observar en una tabla posteriormente. Ethereum genera una cantidad de 18 millones de ether cada año para poder operar en la red, mientras que Bitcoin, por la forma en la que fue diseñado, únicamente tiene un total de 21 millones, lo cual conlleva que, a priori, sea deflacionaria y a largo plazo pueda llegar a perder su valor (aunque hoy día sea un objeto de especulación).

[9] «Cmc markets,» [En línea]. Available: https://www.cmcmarkets.com/es-es/aprenda-a-operar-con-criptomonedas/que-es-ethereum#:~:text=Ethereum%20es%20una%20plataforma%20digital,al%20mercado%20de%20las%20criptomonedas. [Último acceso: 8 Abril 2021].

	BITCOIN	ETHEREUM
NACIMIENTO DE LA PLATAFORMA	18 de Agosto de 2008 (registro del dominio 'Bitcoin.org'). 31 de Octubre de 2008 fecha de su White Paper.	Diciembre de 2013
FECHA 1ER BLOQUE MINADO	3 de Enero de 2009	30 de Julio de 2014
CREADOR DE LA PLATAFORMA	Satoshi Nakamoto, del cual no se sabe quién es o quiénes son (en caso de pseudónimo de una organización)	Vitalik Buterin; Otros co-fundadores incluidos Gavin Wood y Joseph Lubin
FUNCIÓN PRINCIPAL DE LA PLATAFORMA	Sistema de pago descentralizado, rápido y seguro, al igual que su propia moneda.	Plataforma de ejecución de contratos inteligentes y aplicaciones descentralizadas (dApps)
TECNOLOGÍA USADA	Blockchain (Cadena de bloques)	
REDES USADAS	Mainnet (Red principal) y Testnet (Red de prueba)	
ALGORITMO DE SEGURIDAD	SHA2, concretamente SHA256	Ethash, una mezcla de protocolos SHA3
NOMBRE DE LA CRIPTOMONEDA	Bitcoin (BTC)	Ether (ETH)
CANTIDAD MÁXIMA A EMITIR DE CRIPTOMONEDA	21 millones de bitcoin en total, por lo tanto, deflacionaria	18 millones por año, por lo tanto, Inflacionaria
SISTEMA DE MINERÍA	Proof of Work (PoW) o Prueba de Trabajo	
MÉTODO DE RECOMPENSA DE LOS MINEROS	Por validación de bloques	Por validación de bloques, de transacciones y por ejecución de contratos inteligentes
PROCESAMIENTO DE LOS BLOQUES	Cada 10 minutos (600 segundos)	Cada 16 segundos
TAMAÑO DE LOS BLOQUES	1 Mb como máximo	Sin definir, pero muy por debajo de 1 Mb
COSTE DE LAS TRANSACCIONES	Todas por igual	Depende del Gas

Tabla 2: Bitcoin vs Ethereum [22]

[22] «Mi Ethereum,» [En línea]. Available: https://www.miethereum.com/ether/bitcoin-vs-ethereum/#toc13. [Último acceso: 8 Abril 2021].

También se diferencian en los tokens utilizados y en las divisiones de estos para operaciones de coste inferior a 1 ether o 1 bitcoin. A continuación, se muestra una tabla con las distintas divisiones.

UNIDAD Y DECIMALES	BITCOIN	ETHEREUM
1	Bitcoin	Ether
10^{-3}	Milbitcoin	Finney
10^{-3}	Microbitcoin	Szabo
10^{-8}	Satoshi	—
10^{-9}	—	Shannon
10^{-12}	—	Babbage
10^{-15}	—	Lovelaces
10^{-18}	—	Wei

Tabla 3: Tipos de divisiones de tokens [22]

Como se puede apreciar, la división o fracción más pequeña del Bitcoin es el *Satoshi*, que equivale a 1^{-8} bitcoins. Mientras que el ether, debido a su necesidad de flexibilidad para poder crear contratos inteligentes de coste muy bajo, necesita tener un token con un mayor número de posibles divisiones o fracciones. De ahí que sea necesario el *Wei*, que equivale a 1^{-18} ether.

3.3. Ethereum Virtual Machine.

Ethereum es una cadena de bloques que se puede programar, diferenciándose de otras cadenas de bloques en que solo permiten ofrecer un conjunto de operaciones predefinidas. Permite a los usuarios construir sus propias operaciones y aplicaciones por medio de los Smart Contracts. Esto se puede realizar gracias a la máquina virtual de Ethereum, EVM (Ethereum Virtual Machine). Se considera como un sistema Turing Completo, ya que puede gestionar código de cualquier complejidad, y puede ser programado en varios lenguajes de programación existentes, como Solidity, que tiene similitudes con C y que explicaremos en profundidad más adelante.

[22] «Mi Ethereum,» [En línea]. Available: https://www.miethereum.com/ether/bitcoin-vs-ethereum/#toc13. [Último acceso: 8 Abril 2021].

La máquina virtual de Ethereum es considerada el corazón de Ethereum, ya que es el sistema que permite tener conectado todos los nodos y actúa como el motor de la plataforma. El uso de la EVM está presente en todas las plataformas que usen la Blockchain de Ethereum.

En todo momento, la base de datos de Ethereum es actualizada por cientos de nodos conectados en la misma red, ejecutándose sobre el EVM, que contienen los Smart Contracts creados previamente por los usuarios de la red. Por tanto, se puede hacer la similitud de que la red Ethereum es un gran ordenador, formado por ordenadores pequeños distribuidos mundialmente, ejecutando el EVM para evitar las pérdidas de bloques y, por consiguiente, su posible fallo de seguridad.

3.4. Ethereum vs Ethereum Clásico.

Actualmente, tras la bifurcación o fork en julio de 2016 de la cadena de bloques de Ethereum, hay dos líneas de Ethereum activas: Ethereum y Ethereum Clásico. Ambas cadenas de bloques son idénticas hasta el bloque 1.920.000. Esto es debido a que en Junio de 2016 un fallo en el código de la Blockchain causó el robo de aproximadamente 40 millones de dólares en el token ether. Todo esto fue debido al proyecto "The DAO", que es un contrato inteligente en la red, y que lo exponía al robo de tokens. Por tanto, la bifurcación se realizó para devolver a sus dueños los tokens que habían sido robados a causa de la vulnerabilidad de ese contrato inteligente.

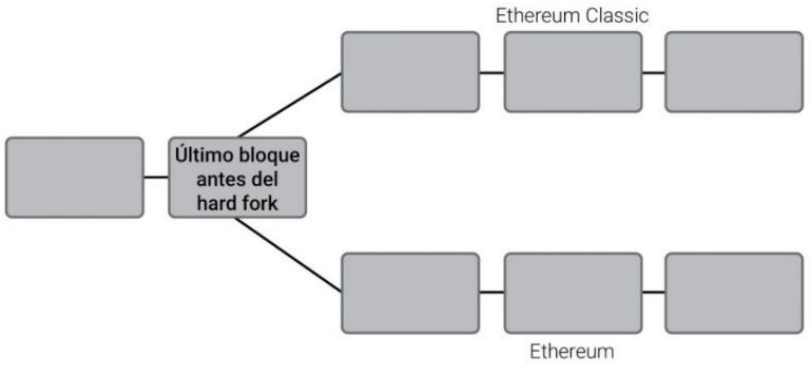

Figura 9: Visualización del forlk y separación de Ethereum y Ethereum Clásico [34]

A continuación, se expone una comparativa entre ambas cadenas de bloques para ver sus diferencias más notables.

[34] S. Lauri, «bitnovo Blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entre-ethereum-y-ethereum-classic/. [Último acceso: 9 Abril 2021].

	Ethereum Clásico	**Ethereum**
Algoritmo de consenso	Utiliza Proof of Work (PoW)	Utiliza Proof of Work (PoW), pero está migrando a Proof Of Stake (PoS)
Cantidad monedas	Tiene un límite de entorno 220 millones de tokens	No tiene un límite establecido

Tabla 4: Comparativa entre Ethereum y Ethereum Clásico [23]

Las ventajas que tiene Ethereum Clásico o Ethereum Classic frente a Ethereum son las siguientes:

- Tiene un alto porcentaje de tecnología heredada del proyecto Ethereum. Esto le permite contar con una poderosa infraestructura para Smart Contracts y DApps, que han sido probadas ampliamente.
- Es un proyecto completamente comunitario. Esto significa que no responde a intereses empresariales o privados, garantizando su descentralización y autonomía.
- Se rige por las premisas de inmutabilidad, no censura y respeto a la privacidad de los usuarios.
- Tiene un camino de desarrollo bien definido que puede impulsar la adopción de su Blockchain. Pese a que es un proyecto joven y ha tenido un inicio accidentado, Ethereum Clásico destaca por sus soluciones.

Las desventajas que presenta Ethereum Clásico o Ethereum Classic frente a Ethereum son las siguientes:

- Carecen de recursos humanos, una situación que ha llevado en algunas ocasiones a un desarrollo lento red la red Blockchain.
- La migración del poder de minería en la red a favor de Ethereum ha puesto en peligro la seguridad de Ethereum Clásico. Muchos analistas comentan que, de seguir esta tendencia, Ethereum Clásico será aún más vulnerable a ataques de 51% como los que han vivido en otras ocasiones. [10] [11]

[10] «bitnovo Blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entre-ethereum-y-ethereum-classic/. [Último acceso: 9 Abril 2021].

[11] «bit2me academy,» [En línea]. Available: https://academy.bit2me.com/que-es-ethereum-classic-etc-criptomoneda/. [Último acceso: 9 Abril 2021].

[23] «bitnovo blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entre-ethereum-y-ethereum-classic/. [Último acceso: 8 Abril 2021].

3.5. Algoritmos de consenso.

Ethereum, al igual que Bitcoin, ha estado utilizando un algoritmo de consenso *Proof of Work (PoW).* Este algoritmo que usa Ethereum se llama Ethash y se basa en descubrir el "nonce" para que el resultado esté por debajo de un determinado umbral de dificultad que marca la red.

Sin embargo, al utilizar una gran cantidad de recursos para descubrir el "nonce", se ha tendido de pasar a utilizar *Proof of Work(PoW) a Proof of Stake(PoS).* Entonces se pasó de *Ethash* a utilizar *Casper*, siendo este último un tipo de algoritmo de consenso basado en PoS.

Figura 10: Icono de Casper [20]

Casper implementa una lógica más compleja, y analiza los nodos de manera que permite identificar cuál es la cadena más larga y beneficiosa para la red. Sus dos características principales son la estabilidad y la información inequívoca. Esto conlleva a que no se pueda generar información contradictoria y garantiza la capacidad de generar un nuevo bloque si, al menos dos terceras partes de las entidades validadoras, lo aprueban.

Esto generará una mayor eficiencia debido a que ya no es necesario ver quién tiene una potencia de cálculo o minado mayor, sino simplemente quién dispone de más recursos para así poder reducir significativamente el tiempo que se tarda en la generación de un nuevo bloque.

Todo esto conlleva una serie de ventajas e inconvenientes que desarrollaremos a continuación:

Ventajas:

- No se basa en poder de cálculo, sino en la participación para un consenso.
- La participación es un mecanismo que se encuentra en el interior de la cadena de bloques, lo que significa que es más fácil diseñar mecanismos para reducir el riesgo de ataque de doble gasto.
- Permite que los proyectos tengan características adicionales como seguridad y descentralización.
- La seguridad se basa en que, si un validador actúa maliciosamente será rápidamente eliminado y castigado, perdiendo todo el ether apostado.

Inconvenientes:

- Poco tiempo de vida, lo cual conlleva que la eficiencia y la seguridad aún no estén comprobadas totalmente.
- No será capaz de finalizar bloques si el sistema de validación se corrompe.
- No es totalmente resistente a ataques del 51%. Si consiguen el poder de validación se corrompería el sistema. [12]

[20] «CRYPTO FOX,» [En línea]. Available: https://crypto-fox.ru/faq/v-chem-sut-protokola-casper/. [Último acceso: 10 Abril 2021].

3.6. Token.

El token utilizado en la red Ethereum es el ether, que es usado para pagar los costes de computación de la máquina virtual de Ethereum.

Ethereum emplea su propio sistema de unidades para el ether. Cada división o fracción se denomina con un nombre diferente, siendo Wei el valor más pequeño y Ether el mayor. Para hacernos una idea de las diferencias de tamaño, 1^18 Weis equivalen a 1 ether.

A continuación, se muestra una tabla con todas las unidades que se corresponden entre Ether y el Wei con sus respectivos valores.

unidad	Valor wei	Wei
Wei	1 wei	1
Kwei (babbage)	10^3 wei	1,000
Mwei (amor)	10^6 wei	1,000,000
Gwei (shannon)	10^9 wei	1,000,000,000
microéter (szabo)	10^{12} wei	1,000,000,000,000
milésimas (finney)	10^{15} wei	1,000,000,000,000,000
éter	10^{18} wei	1,000,000,000,000,000,000

Tabla 5: Unidades de medida de los tokens de Ethereum [24]

Para poder obtener ether se necesita al menos realizar una de estas tareas:

- Convertirse en minero de la red Ethereum.
- Hacer un intercambio de tokens con otro vigente en ese momento. Un ejemplo sería el cambio de bitcoin a ether.
- La compra de ether a través de dinero *fiat*, por medio de alguna API o monedero que incorpore esa funcionalidad.

Ethereum es distinto al resto de redes Blockchain, debido a que su token se utiliza no solo para transferir dinero de una cartera a otra, sino que también se utiliza para pagar el gas que consumen nuestros contratos inteligentes. [13]

[12] «Blockchain Council,» [En línea]. Available: https://www.blockchain-council.org/blockchain/a-quick-guide-to-ethereum-casper/#:~:text=Ethereum%20Casper%20is%20the%20POS%20%28Proof-of-Stake%29%20protocol%20that,two%20co-developed%20Casper%20implementations%20in%20the%20Ethereum%20ecosystem.. [Último acceso: 13 Abril 2021].
[13] «Ethreum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].
[24] «Ethereum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].

3.7. Gas.

El concepto de gas se refiere al coste que conlleva realizar ciertas operaciones dentro de la red. Es la unidad utilizada para medir el trabajo que se realiza en Ethereum. Cada operación realizada en la red tendrá un coste asociado, según la complejidad y el gasto computacional.

Las funciones del Gas son las siguientes:

- Asignar un coste a la ejecución de las operaciones. Cada operación exige una capacidad de cómputo diferente que, dependiendo de los recursos que consuma y su complejidad, tendrá un coste u otro.
- Asegurar el sistema. En el momento que se exige un gasto al realizar cualquier operación, fuerzas a que los integrantes de la red realicen solo las operaciones imprescindibles. Esto repercute en aligerar los bloques, omitiendo la información inútil y evitando la creación de bucles o la introducción de spam, que inutilizaría la red debido a su sobrecarga.
- Recompensar a los mineros por el gasto computacional. Este gasto, que aceptamos al operar en la red, servirá para recompensar a los mineros que han puesto sus recursos a disposición de la red para validar las transacciones. Por lo que resulta un sistema de recompensa e incentivo para mantener la estabilidad de la red.

Aparecen una serie de conceptos derivados del propio gas, que son los siguientes:

- Coste del Gas: es un valor estático que se utiliza para el cálculo del coste de las operaciones en términos de gas. Como el ether fluctúa de valor, es mejor utilizar como medida el gas que cambia menos de valor, y así se consigue que todos los costes sean iguales a lo largo del tiempo.
- Precio del Gas: es el valor del gas en términos de otro token, en este caso el ether. Es un valor menos fluctuante que el ether, aun así, es afectado por los cambios de valor del ether. El precio del gas se establece por el precio de equilibrio de cuánto están dispuestos a gastar los usuarios y cuántos nodos de procesamiento están dispuestos a aceptar.
- Límite del Gas: es el valor máximo de gas que admite un bloque. También se puede ver como la carga computacional máxima que un minero es capaz de procesar. Este valor se puede variar de forma paulatina con el tiempo, si los mineros así lo desean.
- Tarifa del Gas: es la tarifa que hay que pagar para poder ejecutar un contrato, o una función (transacción) dentro de éste en particular. El valor de pago por utilizar las funciones varía dependiendo de la complejidad de éstas. Las tarifas son abonadas a los mineros o a los delegados si se utilizara PoS (Proof of Stake). [14] [15]

[14] «Ethereum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].

[15] «Miethereum,» [En línea]. Available: https://www.miethereum.com/ether/gas/#toc1. [Último acceso: 15 Abril 2021].

3.8. Wallet.

Es un tipo de cartera Bitcoin que permite enviar y recibir criptomonedas de forma totalmente segura gracias a la cadena de bloques.

Para poder realizar estas operaciones es necesario el uso de una clave privada y una clave pública. La clave privada es un conjunto de números generados, de forma aleatoria, de una longitud de 256 bits. La clave pública está generada en base a la clave privada mediante la criptografía de curva elíptica ECDSA.

Como hemos dicho, la Wallet se compone de una clave pública y una clave privada. Para poder realizar cualquier operación es necesaria la combinación de ambas claves. Para acceder a tu Wallet necesitas tu clave privada, que no debe ser guardada en ningún tipo de dispositivo electrónico que sea vulnerable al hackeo. Para guardar la clave existen multitudes de opciones, como programas o aplicaciones que encriptan una parte del S.O, y así únicamente se pueda acceder a través del programa o aplicación. Otro método más popular es guardar la clave privada en un medio físico, siendo este por ejemplo la propia cartera física del usuario.

Un ejemplo de transferencia de dinero sería el siguiente:

"Un inversor recibe bitcoins, por ejemplo, enviando su clave pública del monedero electrónico al vendedor que los deposita en el monedero, y firmando el acto con su clave privada."

Las ventajas que presenta el uso de Wallets, frente al convencional método de envio de dinero a través de bancos u otras entidades, son las siguientes:

- **Control absoluto:** tienes el poder absoluto de tu dinero de modo que, únicamente tú podrás moverlo, y a su vez, tenerlo disponible en todo momento sin posibilidad de que sea retirado por una empresa externa.
- **Anonimato:** no es necesaria la verificación de la cuenta mediante tu identidad, correo electrónico, teléfono u otro medio.
- **Seguridad:** la seguridad ante hackers es extrema, debido a que la cadena de bloques no permite borrar bloques, solo añadir nuevos. Esto, sumado a que se hace una copia en múltiples nodos de la red, imposibilitan el hackeo de las cuentas o Wallets.

Utilizaremos para el desarrollo del proyecto MetaMask, que es una extensión que se ejecuta en el navegador, y que permite acceder a la Wallet a través de la clave privada. [16]

3.9. Smart Contract.

Los contratos inteligentes o Smart Contracts, en realidad, no son "contratos inteligentes" como cabría esperar al leer su nombre. Son programas informáticos con una serie de condiciones de partida que garantizan el cumplimiento de un acuerdo de forma automática, transparente y segura cuando las condiciones necesarias se hayan cumplido.

[16] «Blockchain Economía,» [En línea]. Available: https://www.blockchaineconomia.es/wallet-blockchain/. [Último acceso: 16 Abril 2021].

Estos contratos inteligentes se utilizan en un ambiente no controlado por ninguna de las partes implicadas en el contrato, puesto que corren en un ambiente descentralizado al haber sido previamente desplegados en la red Blockchain.

La principal diferencia de un contrato tradicional a un contrato inteligente es que, este último, está escrito en un lenguaje de programación y no tiene diferentes formas de interpretarlo dependiendo del interés de cada momento.

Para entenderlo mejor, se expondrá un ejemplo sencillo. Imaginemos que una persona le tiene que dar una cantidad **X** de dinero a otra persona el día **Y**. Entonces ambas partes redactan el contrato inteligente y lo despliegan. Cuando llegue el día **Y**, de la cuenta saldrá **X** dinero hacia la otra cuenta de la persona que redactó el contrato. Como acabamos de ver por medio del ejemplo, ninguna de las dos partes interactúa nuevamente una vez desplegado el contrato, y no ha sido necesaria la figura de ningún notario o agente externo que estuviera presente para poder hacer la transacción, ahorrando costes.

Los Smart Contracts presentan una serie de ventajas, frente a los contratos tradicionales, y son las siguientes:

- **Autonomía**: este tipo de contratos se dan entre varias entidades, sin ningún intermediario de por medio. No es necesario que por ejemplo un abogado valide el acuerdo. De esta manera se reducen, de manera considerable, el número de agentes externos que estén implicados en la creación del contrato.

- **Confianza**: al utilizar la tecnología Blockchain todos estos contratos están copiados en cada nodo y van directos a la cadena de bloques. Esto significa que los contratos están encriptados, de modo que solo las personas que formen parte del acuerdo puedan leerlo.

- **Velocidad: a**umenta la velocidad en los procesos de negocio. Los Smart Contracts no dejan de ser código que se ejecuta de manera automática, y no necesita ningún tipo de interacción humana que pueda ralentizar el proceso, cosa que no ocurre con los contratos tradicionales.

- **Costes:** los costes son más reducidos porque no necesitan ningún agente externo, como un notario para validar el contrato.

- **Seguridad:** estos contratos que se despliegan en la red Blockchain se guardan en la cadena de bloques, de manera que no pueden perderse ni pueden modificarse sin alterar toda la cadena. Recordamos que todo cambio en la cadena de bloques queda registrado, y en cualquier momento se puede comprobar el cambio que ha sufrido.

- **Nuevos modelos de negocio:** los contratos inteligentes permiten nuevos tipos de negocios que abren nuevas vías de innovación y emprendimiento.

Hay que recordar que cada operación que se encuentre dentro de un Smart Contract tiene un coste de gas asociado. A continuación, podemos ver una tabla con todas las posibles operaciones que se pueden realizar, con los costes que estas generan.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.
$G_{quaddivisor}$	100	The quadratic coefficient of the input sizes of the exponation-over-modulo precompiled contract.

Figura 11: Tabla de gastos de operaciones [21]

Se puede observar cómo la tabla está dividida en tres columnas. La primera columna corresponde al nombre de la operación a realizar. La segunda columna corresponde al valor en gas que consume realizar la operación. La tercera columna corresponde con una pequeña descripción de la operación que se realiza para un mayor entendimiento. Estos valores de coste pueden modificarse, como se explicó previamente en el apartado de Gas.

[21] «Mi Ethereum,» [En línea]. Available: https://www.miethereum.com/ether/gas/#toc1. [Último acceso: 18 Abril 2021].

CAPÍTULO 4

Tecnologías utilizadas

A continuación, tras haber explicado en qué consiste Blockchain y más en concreto Ethereum, vamos a exponer las tecnologías que han sido necesarias a la hora de desarrollar el siguiente proyecto.

4.1. VMware.

Figura 12: Icono de WMware [35]

WMware es un sistema que nos permite emular un sistema operativo. En nuestro caso: Ubuntu 20.04 LTS. Este nos permite simular, a todos los efectos, un computador físico con

[35] «cibercafe,» [En línea]. Available: http://modvsub1-2.blogspot.com/2016/05/vmware-workstation-pro.html. [Último acceso: 20 Abril 2021].

CPU, BIOS, tarjeta gráfica, memoria RAM, tarjeta de red, sistema de sonido, conexión USB y disco duro, entre otros aspectos.

Un dato negativo de la utilización de un sistema operativo que funciona en el hardware emulado es que la velocidad de ejecución es menor, pero en este caso es suficiente.

4.2. Visual Studio Code.

Figura 13: Icono de Visual Studio Code [36]

Visual Studio Code es un IDE (Entono de Desarrollo Integrado), utilizado para la programación y edición de código necesario para la implementación de los Smart Contracts y para la programación del entorno virtual de la página Web. Dicha página será la interfaz gráfica que utilizaremos para poder interactuar con los distintos contratos desplegados en nuestra red.

Se ha elegido el uso del *IDE Visual Studio Code* debido a que puede incorporar distintas características extras dependiendo de las necesidades del programador en cada momento. En este caso, solo ha sido necesario instalar una extensión denominada *Solidity,* debido a que por defecto el resto de los lenguajes de programación ya son reconocidos por el IDE. Solidity, como se va a explicar posteriormente, es el lenguaje utilizado para la generación de Smart Contracts. Este lenguaje es interpretado por la EVM que gestiona los contratos de forma automática.

4.3. Extensión de Solidity.

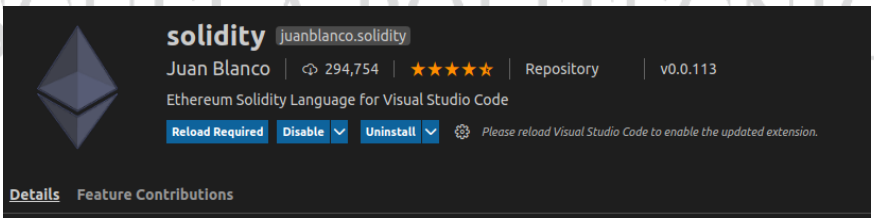

Figura 14: Extensión de Solidity para Visual Studio Code

Esta extensión, creada por Juan Blanco, permite al IDE reconocer el lenguaje de Solidity, además de poder compilar los contratos y tener funciones de autocompletado.

Lo interesante de esta extensión es que, si la sintaxis es errónea, te lo muestra para que puedas corregirla.

[36] «DEV,» [En línea]. Available: https://dev.to/glaucia86/14-tips-para-optimizar-su-visual-studio-code-40ia. [Último acceso: 20 Abril 2021].

Esta extensión solo se ha utilizado cuando ha habido que añadir algunas variables o funciones a los contratos. En el momento de generar los contratos enteros desde el inicio, se ha recurrido al uso de Remix, que es más completa en ese aspecto.

4.4. REMIX.

Figura 15: Icono de Remix IDE [37]

Remix es un IDE que se puede ejecutar de forma online u offline y se utiliza para poder realizar programas con el lenguaje de alto nivel Solidity.

Se ha elegido la utilización de este IDE por la accesibilidad de poder acceder a este mediante un navegador web, ya que es una herramienta enfocada en la programación con Solidity.

Las características de este IDE, que incorpora respecto a otros, son las siguientes:

- **Compilación:** este IDE nos permite compilar nuestros contratos programados con Solidity con distintas versiones del compilador. Esto es una característica a tener en cuenta debido a que el lenguaje es relativamente nuevo y permite que, funciones que en una versión funcionan de forma correcta, dejen de hacerlo en versiones posteriores. También hay que destacar el uso de nuevas variables que han aparecido en versiones más actuales.

- **Corrección de errores:** dependiendo de la versión de compilador seleccionada, nos muestra los errores que presentan los Smart Contract y nos orienta sobre las actuaciones a realizar en el contrato para su correcto funcionamiento.

- **Despliegue de contratos:** nos permite el despliegue de contratos en una red de prueba que se crea en el propio IDE. Esto nos ayuda a poder comprobar un correcto funcionamiento de los contratos una vez desplegados.

[37] «Github,» [En línea]. Available: https://github.com/ethereum/remix-ide/releases. [Último acceso: 21 Abril 2021].

4.5. MetaMask.

Figura 16: Icono de MetaMask [38]

MetaMask es un plugin que se instala en el navegador, y hace de puente entre el navegador y las aplicaciones descentralizadas de Blockchain, permitiendo interactuar en la red con total seguridad y con la posibilidad de gestionar varias Wallets de forma simultánea.

Este plugin nos permite firmar transacciones, gestionar tokens y administrar diferentes Wallets, todo esto a través de una interfaz fácil que agrupa a todos los servicios que ofrece Ethereum.

Lo mencionado anteriormente es posible gracias a la integración del API Web3 en el JavaScript de cada sitio Web.

MetaMak, además, proporciona una capa extra de seguridad debido a que posee su propia llave privada de manera que, en vez de manejar las Wallets, maneja los tokens propios haciendo que las transacciones nunca modifiquen la configuración del servidor y evitando así riesgos innecesarios. [17]

4.6. Node.js

Figura 17: Icono de Node.js [39]

Node.js es un entorno de ejecución de JavaScript orientado a eventos asíncronos. Su utilización está pensada para la creación de aplicaciones Network escalables. Lo interesante de node.js es que incluye *Node Package Manager (NPM),* un gestor de paquetes que vamos a utilizar para la

[17] «Criptonoticias,» [En línea]. Available: https://www.criptonoticias.com/aplicaciones/metamask-puente-ethereum-navegador/. [Último acceso: 20 Abril 2021].
[18] «nodejs,» [En línea]. Available: https://nodejs.org/es/about/. [Último acceso: 20 Abril 2021].

[38] «Metamask,» [En línea]. Available: https://metamask-devcon4.herokuapp.com/#slide=1. [Último acceso: 22 Abril 2021].

[39] «Node js,» [En línea]. Available: https://download.logo.wine/logo/Node.js/Node.js-Logo.wine.png. [Último acceso: 22 Abril 2021].

instalación y gestión de distintos paquetes y librerías, necesarios para el correcto funcionamiento de nuestro proyecto. [18]

4.7. React.

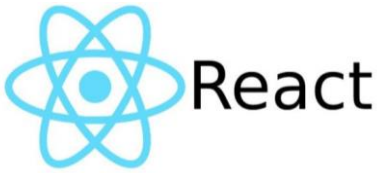

Figura 18: Icono de React [40]

React es una biblioteca GUI de código abierto de JavaScript que se centra en el desarrollo de interfaces de usuario. Este tiene un completo ecosistema de módulos, herramientas y componentes capaces de ayudar al desarrollador. Los módulos, que han sido necesario instalar, se comentarán más adelante.

Una característica que trae React es el uso de *props* y *states*.

- Los *states* son variables que se utilizan sobre el código que corresponde a una página Web y se pueden considerar variables globales dentro del código de esa página o componente.
- Los *props* son como los states, pero estos vienen de una estancia superior; es decir, de un componente padre y se utiliza para poder interactuar con ese valor dentro del componente hijo. Esta funcionalidad es muy útil, debido a que el valor de una variable se puede utilizar en otros componentes siendo necesario, únicamente, la utilización de los states y los props.

A continuación, vamos a visualizar una tabla en la que se pueden ver las diferencias entre props y state más claramente.

	Props	State
¿Puede tener un valor inicial desde el componente padre?	Si	Si
¿Puede ser cambiado por el componente padre?	Si	No
¿Puede tener valores por defecto?	Si	Si
¿Puede cambiar dentro de un Componente?	No	Si
¿Puede tener un valor inicial para el componente hijo?	Si	Si
¿Puede cambiar en el componente hijo?	Si	No

Tabla 6: Comparativa entre State y Props [25]

Como se puede apreciar en la tabla anterior, los props al venir de un componente padre no pueden ser modificados en su valor, siendo totalmente distinto con una variable tipo state.

[25] «Github,» [En línea]. Available: https://github.com/uberVU/react-guide/blob/master/props-vs-state.md. [Último acceso: 20 Abril 2021].

[40] N. Birch, «onexxtrapixel,» [En línea]. Available: https://onextrapixel.com/increase-knowledge-of-react-js-with-these-helpful-resources/. [Último acceso: 23 Abril 2021].

Esta característica, aunque parezca de poca importancia, es la que hace más interesante el uso de este lenguaje.

Otra de las características más importantes que tiene el uso de React es la posibilidad de visualización de variables, indistintamente del tipo que sean, mediante el uso de llaves **{}** y permitiendo así un ahorro de tiempo considerable.

4.8. Bootstrap.

Figura 19: Icono de Bootstrap[43] [41]

Bootstrap es un framework CSS empleado para desarrollar aplicaciones y sitios Web. Su implementación consigue complementar el entorno visual que estará cara al usuario. Tiene un sistema que permite dividir la pantalla en cuadriculas y, posteriormente, insertar el contenido en columnas para poder hacerlo atractivo visualmente a través de pantallas de distintos tamaños. Contiene numerosas plantillas con elementos de navegación basados en HTML, CSS y JavaScript.

Para poder utilizar todas las herramientas y opciones que nos trae Bootstrap, es necesario declarar previamente en el código del componente cuáles son las que pretendemos utilizar.

4.9. Truffle Suite.

Truffle Suite es un **conjunto de herramientas** que permiten a los desarrolladores realizar las pruebas de sus aplicaciones descentralizadas de forma sencilla y eficiente.

Se trata de un entorno de desarrollo que permite poder observar y controlar los Smart Contracts en todo su ciclo de vida; es decir, desde su creación hasta su despliegue en la red Ethereum o en otras redes que utilicen el EVM.

Las herramientas de que dispone Truffle Suite son las siguientes: *Ganache, Truffle, Drizzle* y las *Truffles Boxes*. Estas últimas no se desarrollarán en este apartado debido a que se explicaran más adelante.

[41] «freepikpsd,» [En línea]. Available: https://freepikpsd.com/bootstrap-icon-png/314535/. [Último acceso: 23 Abril 2021].

4.9.1. Ganache.

Figura 20: Icono de Ganache [42]

Ganache es una aplicación que nos permite, de forma rápida, desplegar una red Blockchain de prueba. Esta red de prueba dispone de diez cuentas, con un total de mil ether divididos entre todas las cuentas de forma ponderada. Para poder hacer uso de las cuentas, Ganache nos permite ver las direcciones de las Wallets, las claves privadas para acceder a estas y las transacciones realizadas. También nos permite desplegar contratos dentro de su red y nos da la posibilidad de poder visualizar el contenido de todos los bloques, que se encuentran en la red de bloques, generados hasta la fecha.

Para poder emplear esta herramienta, sacándole el máximo partido, la utilizamos a través de MetaMask. Esto nos permitirá poder intercambiar cuentas de forma más eficiente y de esa manera optimizar el tiempo. Para poder interactuar con Ganache, solo es necesario copiar la clave privada de la Wallet en cuestión. Por parte de Metamask, primero se tiene que conectar a la red creada con Ganache y, posteriormente, introducir la clave privada para poder acceder a la cuenta.

Los pasos mencionados se expondrán a continuación de forma visual.

- De Ganache seleccionamos una cuenta de las diez que hay disponibles y copiamos su clave privada o Private Key (Figura 21).

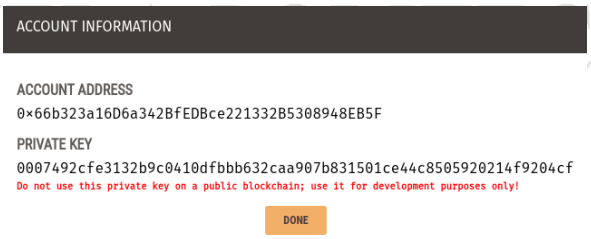

Figura 21: Información de una cuenta de Ganache

- Por último, en la parte que corresponde a la interacción con Ganache, hay que tener en cuenta qué puerto está siendo utilizado para su funcionamiento (Figura 22)

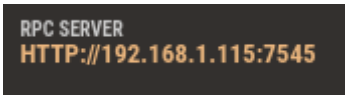

Figura 22: Dirección donde se aloja Ganache

[42] «Truffle Suite,» [En línea]. Available: https://www.trufflesuite.com/docs/ganache/overview. [Último acceso: 24 Abril 2021].

- Usando el plugin, si aún no está configurada MetaMask para conectarse a la red desplegada con Ganache, se configura como se muestra a continuación, introduciendo en el campo *New RPC URL* la dirección y puerto donde se encuentra corriendo Ganache (Figura 23).

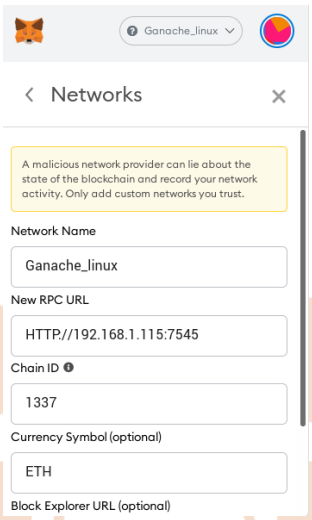

Figura 23: Configuración de MetaMask

- Una vez configurado para su conexión con la red desplegada, seleccionaremos *importar Wallet* para poder introducir la clave privada copiada previamente (Figura 24)

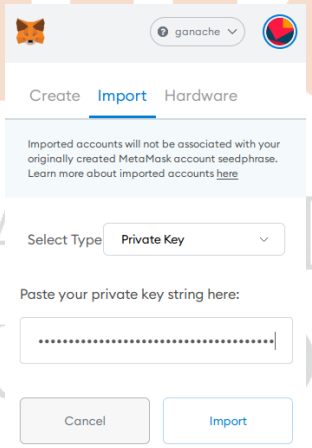

Figura 24: Importar cuenta a MetaMaask

- Una vez realizado todos los pasos, ya se puede acceder a la Wallet de Ganache por medio de MetaMask y así empezar a realizar pruebas (Figura 25).

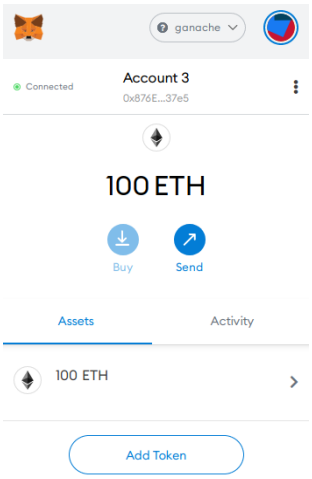

Figura 25: Cuenta de MetaMask enlazada

4.9.2. Truffle.

Figura 26: Icono de Truffle [43]

Es una herramienta de desarrollo capaz de probar e implementar proyectos que permite desplegar en su propia red de prueba (Ganache) y posibilitando la compilación, enlazado y despliegue de los Smart Contracts. También admite el testeo automático de Smart Contracts. Todo ello nos permite visualizar, mediante una consola, todos los procesos por el que pasa el contrato para su despliegue; es decir, su compilación y, después, su migración. Truffle nos posibilita migrar nuestros Smart Contracts tanto a redes privadas como a redes públicas.

4.9.3. Drizzle.

Figura 27: Icono de Drizzle [44]

[43] «Github,» [En línea]. Available: https://github.com/trufflesuite. [Último acceso: 24 Abril 2021].
[44] «npm,» [En línea]. Available: https://www.npmjs.com/package/@drizzle/store. [Último acceso: 24 Abril 2021].

Drizzle es una colección de bibliotecas de front-end utilizada para la interacción con Smart Contracts. El núcleo de Drizzle es de Redux; es decir, tiene todas las características que se utilizan con React: los props y los states. Se utiliza Drizzle y no otro, debido a que mantiene la sincronización constante de los datos del contrato con las transacciones realizadas, entre otros aspectos.

Como se puede ver en la imagen de abajo, Drizzle se basa en módulos para poder complementarse.

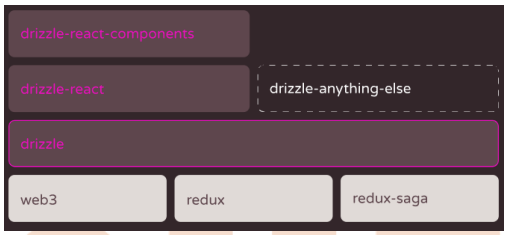

Figura 28: Módulos de Drizzle [47] [45]

Los módulos que incorpora son los siguientes:

- **drizzle**: la biblioteca principal responsable de la creación de instancias de web3, cuentas y contratos, conectando las sincronizaciones necesarias y proporcionando funcionalidad adicional del contrato.
- **drizzle-react:** proporciona un componente *DrizzleProvider* y un método auxiliar *drizzleConnect* para facilitar la conexión de Drizzle con su aplicación React.
- **drizzle-react-components**: una biblioteca de componentes útiles para funciones Dapp comunes. Actualmente incluye *ContractData, ContractForm y LoadingContainer*. [19]

4.10. Solidity.

Figura 29: Icono de Solidity [46]

Solidity es el lenguaje de programación más popular a la hora de la creación de Smart Contracts para Ethereum. Es un lenguaje de alto nivel parecido a Javascript diseñado para correr en la máquina virtual de Ethereum (EVM). Se considera como un lenguaje de tipo 'Turing Completo', debido a que puede resolver cualquier tipo de cómputo y es capaz de añadir códigos más complejos, como por ejemplo bucles. Recordamos que para programar con este lenguaje hemos utilizado *Remix* y *Visual Studio Code* como IDE´s para poder trabajar de forma cómoda.

[19] «TRUFFLE SUITE,» [En línea]. Available: https://www.trufflesuite.com/drizzle. [Último acceso: 24 Abril 2021].
[45] «Truffle Suite,» [En línea]. Available: https://www.trufflesuite.com/drizzle. [Último acceso: 24 Abril 2021].
[46] «ANQUES TECNOLABS,» [En línea]. Available: https://www.anques.com/top-solidity-development-company/. [Último acceso: 24 Abril 2021].

Hay que destacar que, como todo lenguaje de programación, tiene palabras reservadas para la realización de ciertas funciones, por ejemplo, la palabra "contract" se utiliza para la inicialización de un nuevo contrato.

A continuación, vamos a hacer una breve explicación de los tipos de variables que permiten utilizar Solidity y los tipos de datos que admite.

Los tipos de datos, que Solidity acepta para poder interactuar con ellos, son los siguientes:

- Enteros: se utiliza para almacenar números enteros. Existen dos tipos:
  - El *uint* almacena números positivos o cero y puede almacenar hasta 256 bits, aunque se puede almacenar un número menor si los definimos como *uint8, uint16, uint 32, uint64 o uint128*. El rango que tiene de almacenamiento es [0 y 4294967295].
  - El *int* almacena tanto valores negativos como positivos. El rango que tiene de almacenamiento es [–2147483648 y 2147483647].

- Bool: puede tener valor True o False únicamente.

- Address: permite almacenar la dirección de una cuenta de Ethereum. Pueden incorporar, a continuación de a*ddress,* la opción *payable* que permite transmitir tokens.

- Byte: representa el tamaño fijado de un array.

- Enum: son enumeraciones que almacenan un valor constante.

- Array: es un tipo de dato estructurado que permite almacenar un conjunto de datos que sean del mismo tipo. Existen dos tipos: los estáticos y los dinámicos.
  - Estáticos: tienen un tamaño fijo y se especifica a la hora de la creación
  - Dinámicos: no tienen un tamaño fijo y se va ajustando de manera dinámica.

- Structs. conjuntos de datos no homogéneos que son definidos por el usuario.

- String: secuencia de caracteres utilizados usualmente para almacenar cadenas de caracteres.

- Mapping: tienen gran similitud a las tablas hash porque almacenan pares de clave-valor.

- Storage: memoria global que está disponible para todas las funciones de un Smart contract. Es un almacenamiento permanente.

- Memory: memoria local que está disponible en todas las funciones de un Smart contract. Es un almacenamiento volátil, debido a que su tiempo de vida se basa en la duración de la función que esté utilizando este tipo de dato.

Una vez explicado los distintos tipos de datos que acepta Solidity, solo nos queda abordar los tipos de funciones que vamos a necesitar a la hora de realizar los Smart Contracts.

Las funciones son de los siguientes tipos:

- Public: las funciones públicas pueden ser llamadas de manera interna o a través de mensajes externos. Se pueden heredar de otros contratos.

- Private: las funciones privadas y el estado de las variables son únicamente visibles internamente; es decir, desde el propio contrato y no desde contratos heredados o derivados.

Cabe destacar que se le puede añadir la palabra *View* a las funciones que no modifiquen ninguna variable del contrato; es decir, las funciones que tienen un gasto de Gas 0. Esto se hace para poder indicar qué funciones no tienen ningún gasto.

Una vez explicadas todas las aplicaciones y tecnologías necesarias, a fin de conseguir los conocimientos esenciales para poder realizar el proyecto, procederemos a explicarlo.

CAPÍTULO 5

Estructura del sistema

5.1. Funcionamiento.

La aplicación que se va a desarrollar deberá interactuar con la red Blockchain para permitirnos gestionar una empresa de *rent a car*. La herramienta necesita cumplir la operatividad de cualquier portal web de este sector, que son las siguientes:

- Permitir crear diferentes oficinas de alquiler (*sucursales*) o sedes por todo el mundo con total libertad. En la herramienta que vamos a desarrollar, nos vamos a centrar en el espacio europeo, aunque en futuras versiones se podría ampliar a otras zonas geográficas.

- Permitir a los administradores registrar los vehículos en todas las sedes, sin tener que estar limitados a la gestión de una sola. Se promueve así la descentralización en la gestión del sistema ahorrando costes, y teniendo presente que quedará identificada y reflejada en la Blockchain las acciones que genere cada administrador.

- Alquiler, devolución y pago, a través de la Web, con la identificación clara y operaciones seguras desde la cartera (Wallet) de los clientes.

- Registro de los clientes para poder alquilar los diferentes vehículos y así estar cubiertos en caso de accidentes. Este punto es importante porque, debido al

anonimato de las Wallets, no se podría relacionar la persona que realizó la acción de alquilar el vehículo con la Wallet asociada.

- Alquilar o entregar el vehículo en el lugar que más le convenga al cliente. Esto genera un gran atractivo y otorga gran libertad a los usuarios, ya que no tienen por qué entregarlo en la misma sucursal donde se recogió, sino que pueden devolverlo en cualquier lugar del territorio europeo.

5.2. Desarrollo del proyecto. Estructura:

Para conseguir todo lo expuesto en el apartado anterior, este trabajo deberá cumplir los siguientes requisitos:

- Desplegar una red Blockchain privada, en nuestro caso Ethereum, que permita utilizar redes de simulación para desarrollar el proyecto piloto, consiguiendo con ello eludir los costes que requiere hacerlo en la red pública real.
- Grabar un registro de todos los solicitantes de los servicios de alquiler en la cadena de bloques de la red anteriormente desplegada.
- Automatizar la concesión del alquiler del vehículo a través de un Smart Contract.
- Desarrollar una función que permita pagar el alquiler, incluso con criptomonedas legales en circulación, dejando registro único de estos pagos.
- Dejar constancia del lugar de inicio y final exacto del alquiler, a través de las coordenadas GPS (se accederá con ellas a Google Maps través de un código QR), que serán guardadas también en la cadena de bloques.
- Diseñar un portal web que dé cabida a gestionar todas las funcionalidades anteriores.

A continuación, se abordará cómo interaccionan las tecnologías expuestas en el capítulo anterior para hacer esto posible.

5.3. Esquema de las tecnologías.

Para el desarrollo del proyecto, es necesario la implementación de las tecnologías descritas en con anterioridad, y la interconexión entre ellas que nos permita conectar la aplicación con las cuentas de cada usuario, de manera que las transacciones queden firmadas y sean confirmadas.

Para poder desarrollar todo lo descrito, hemos divido el proyecto en tres partes claramente diferenciadas.

- Smart Contracts: se explicarán todos los contratos necesarios que podrán ser desplegados en nuestra red de prueba *Ganache*. Para realizar este apartado, ha sido necesario el uso de los *IDE Remix* y *Visual Studio Cod*e. La utilización de *Remix* ha sido clave, debido al debugger que incorpora y a su facilidad a la hora de desplegar contratos para comprobar su correcto funcionamiento.

- Interfaz usuario: se explicarán todos los programas y lenguaje utilizados para poder hacer posible la interfaz de usuario, de forma que responda a las necesidades de estos. Tendrá la posibilidad, desde una cuenta de Administrador, de generar sucursales y

vehículos enlazados a cada una de ellas. Por parte del usuario, tendrá la posibilidad de alquilar y devolver el vehículo de forma fácil y sencilla, pagando por el tiempo de uso.

- Interconexión: Esta parte es la más importante ya que permitirá interactuar a la Web con los contratos desplegados en la red de *Ganache*. También permitirá a los usuarios enlazar la Wallet con el navegador Google Chrome para poder interactuar con nuestro portal Web.

A continuación, vamos a observar de forma visual cómo interactúan todas las tecnologías descritas anteriormente.

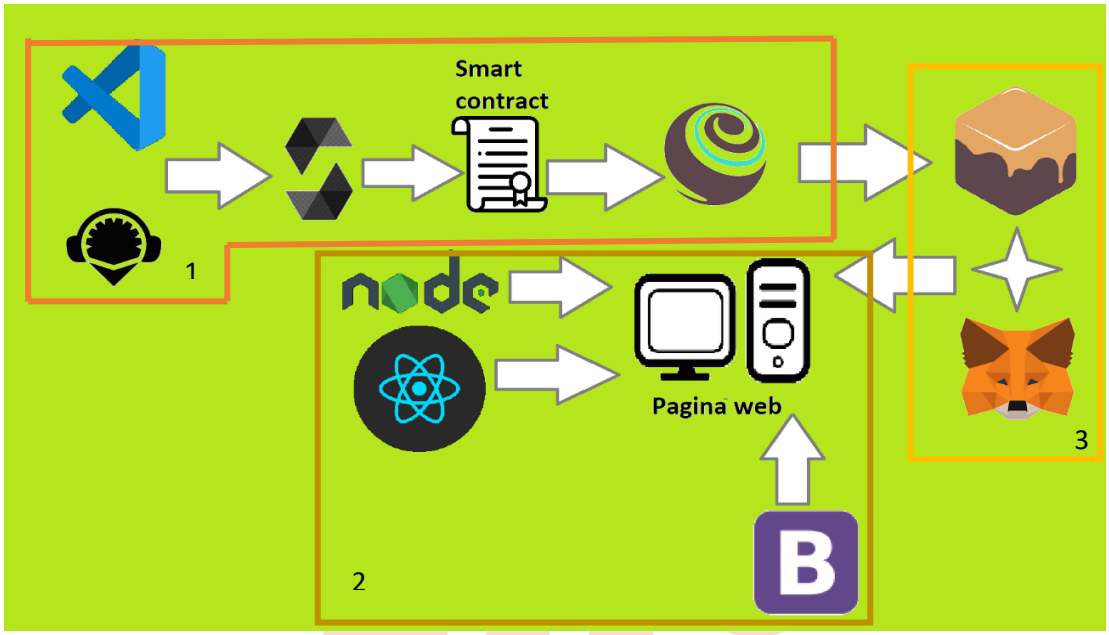

Figura 30: Implementación de las tecnologías de forma visual

En el esquema anterior, se puede comprobar la estructura que tendrá la red y cómo se relacionan los elementos unos con otros, posibilitando así la buena ejecución de nuestra aplicación.

Se pueden apreciar las tres zonas mencionadas anteriormente, siendo el apartado de Smart Contract la correspondiente al recuadro número 1; la interfaz de usuario utilizando *React, NodeJs* y *Bootstrap* con el número 2; la de interconexión, mediante *MetaMask* y *Ganache* con el número 3.

Desarrollando un poco más las divisiones creadas en el esquema visual superior, en el recuadro con el número 1 podemos apreciar cómo, mediante *Visual Studio Code* y mediante *Remix*, programamos con el lenguaje *Solidity* los Smart Contracts. Posteriormente mediante *Truffle*, desplegamos los contratos en la red *Ganache*.

Con el recuadro número 2, se puede observar cómo mediante *NodeJs, React* y *Bootstrap* creamos la interfaz de usuario de la Web. Como se ha mencionado en el apartado de "Tecnologías utilizadas", utilizamos *NodeJs* a fin de instalar los paquetes necesarios para el correcto funcionamiento de la página Web. Los otros dos se utilizan para la programación de dicha página.

En el recuadro número 3, se observa cómo está incluido *MetaMask* y *Ganache*. Para poder acceder desde nuestra Web a la red Blockchain para pruebas (*Ganache*) deberemos utilizar la herramienta *Metamask* desde el cliente Web que queramos emplear. Se ha configurado *Ganache* (con IP estática) para poder ser utilizada desde cualquier equipo de la red y no solamente desde el PC donde se está ejecutando.

5.4. Diseño.

En este punto se desarrollará la estructura del proyecto creado y se mostrarán los esquemas de flujo para la explicación de las posibles interacciones.

El esquema del proyecto es el siguiente:

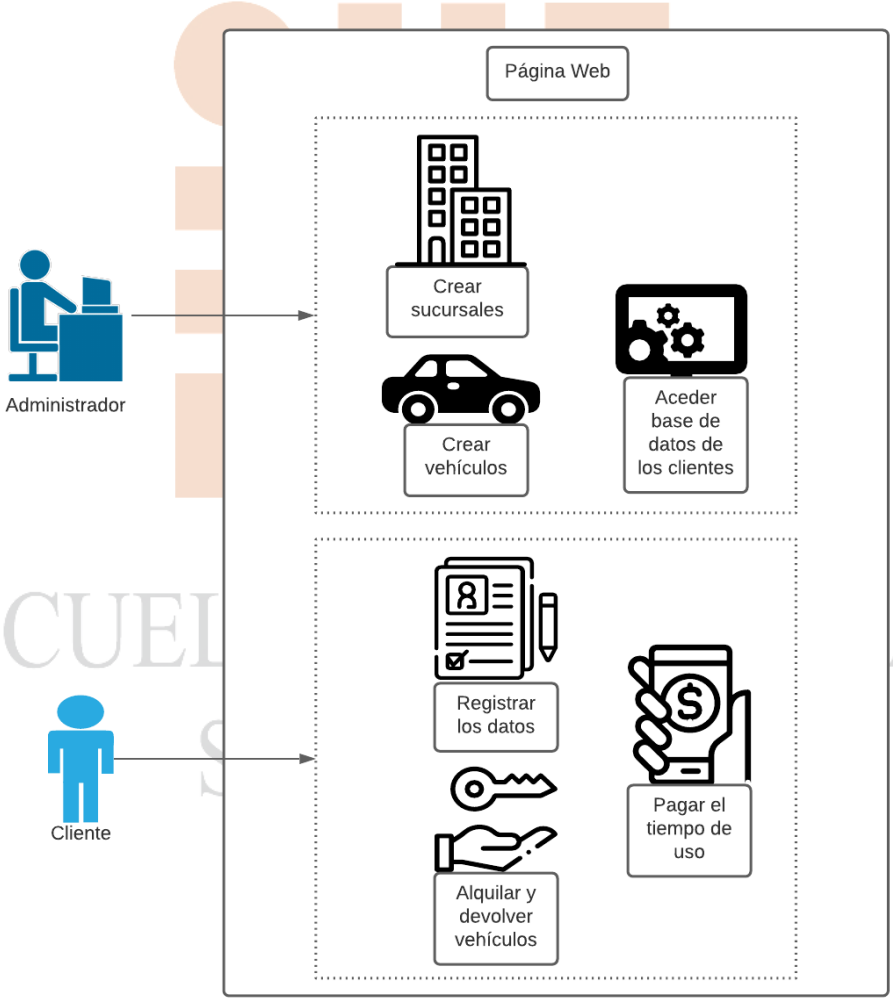

Figura 31: Esquema de funcionamiento del proyecto

Como se puede apreciar en la imagen superior, las figuras que tenemos como “Administrador” y como “Cliente” son los distintos tipos de agentes que se contemplan en el proyecto.

La figura de “Administrador” es la encargada de generar tanto las sucursales como los vehículos que se encuentran en ésta. Por otra parte, debido a su posición de poder dentro de

la jerarquía que se ha establecido, tiene la posibilidad de acceder a la base de datos donde se aloja toda la información de los clientes que se han registrado en el sistema.

La figura de "Cliente" tiene la posibilidad de registrarse en el sistema si quiere acceder a los servicios que ofrece la empresa. Si se registra podrá tanto alquilar los vehículos como devolverlos, y todo ello desde la propia página Web.

Una vez visto, a grosso modo, cuáles son los posibles agentes que pueden interactuar con el sistema vamos a centrarnos en las posibles interacciones de estos:

5.4.1. Creación de sucursales.

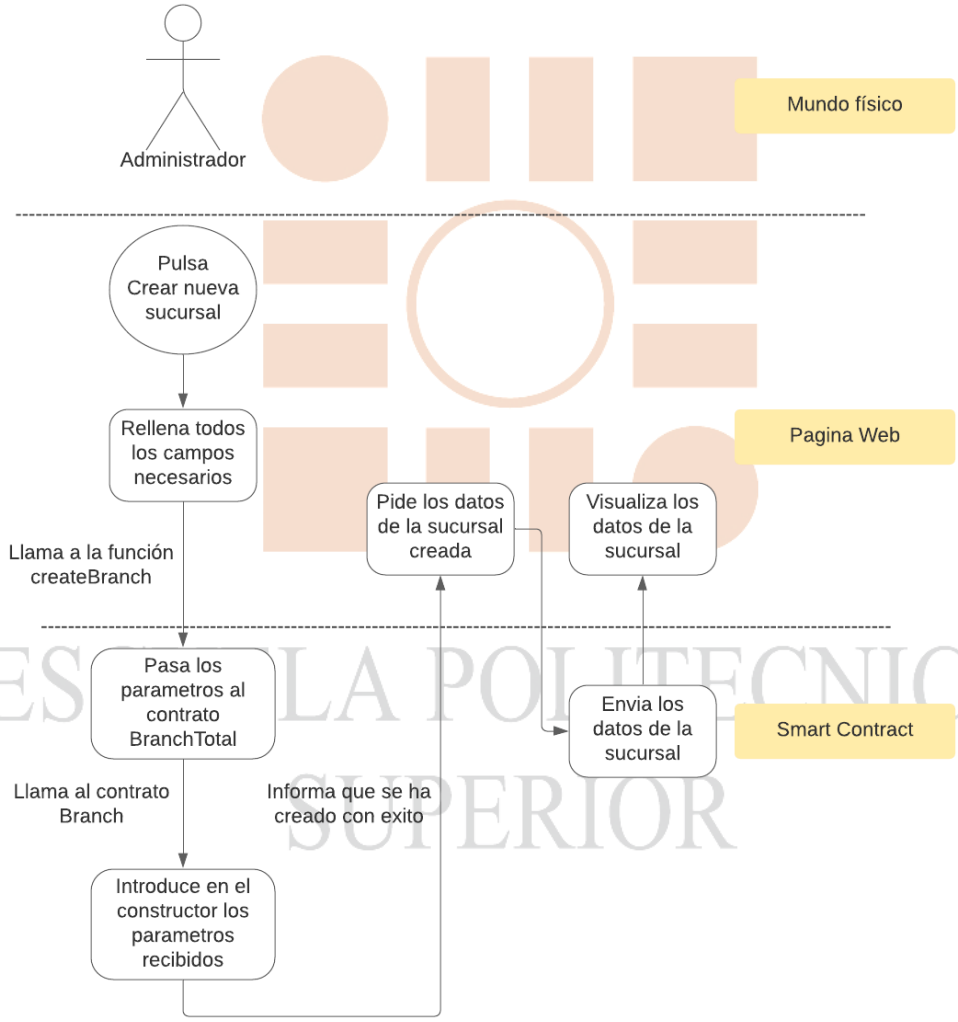

Figura 32: Diagrama de secuencia de creación de sucursales

En este diagrama de flujo podemos observar cómo para la creación de sucursales solo se necesita, como requisito, ser un administrador del sistema.

Los pasos a desarrollar son los siguientes:

1. El administrador decide crear una nueva sucursal y pulsa sobre el botón de “Crear nueva sucursal”.
2. A continuación, le aparecerá un desplegable con una serie de campos que tendrá que cumplimentar de forma correcta para poder generar la sucursal de forma exitosa. Los campos que tiene que rellenar son: *Nombre, Localización, Wallet de la empresa* (se suele introducir la del administrador que crea la nueva sucursal) e *información de interés*.
3. Una vez hecho, pulsará el botón “Crear nueva sucursal” y esta llamará a la función *createBranch*, que se encuentra en el contrato *BranchTotal*.
4. La función *createBranch* pasará los parámetros al constructor del contrato *Branch* y este creará una nueva sucursal con todos los datos recibidos.
5. Informará de su creación de forma exitosa aumentando, en una unidad, la cantidad de sucursales creadas.
6. La página Web solicitará la información de la nueva sucursal creada.
7. El contrato **Branch** le pasará los datos que le solicitan.
8. Por último, en la página Web se podrán visualizar todas las sucursales creadas de forma satisfactoria.

Esta es una descripción básica de los pasos necesarios para la creación de sucursales. En posteriores apartados, explicaremos de forma más detallada las funciones que han interactuado para realizar esta tarea.

5.4.2. Creación de vehículos:

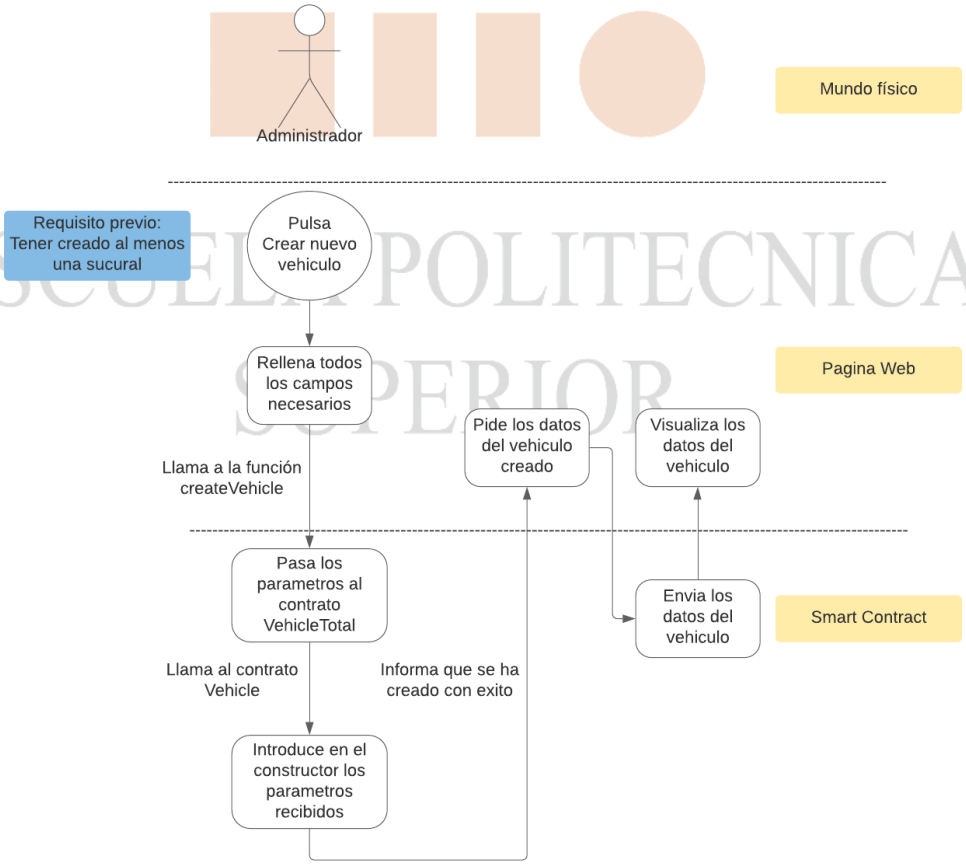

Figura 33: Diagrama de secuencia de creación de vehículos

En este diagrama de flujo, se puede apreciar cómo se desarrolla la interacción para la creación de un nuevo vehículo. El único requisito previo es tener creado al menos una sucursal. Esto es necesario debido a que, de lo contrario, no se pueden enlazar los vehículos a sus sucursales.

Los pasos a desarrollar son los siguientes:

1. El administrador decide crear un nuevo vehículo y pulsará sobre la sucursal donde quiere introducir los datos de este.
2. A continuación, le aparecerá un desplegable con una serie de campos que tendrá que rellenar de forma correcta para poder generar el vehículo de forma exitosa. Los campos que se deben cumplimentar son: *Matrícula del vehículo, Nombre de la localización, Localización y tipo de vehículo.*
3. Una vez hecho, pulsará el botón "Crear nuevo vehículo" y se llamará a la función *createVehicle*, que se encuentra en el contrato **VehicleTotal**.
4. La función *createVehicle* pasará los parámetros al constructor del contrato **Vehicle** y este creará un nuevo vehículo con todos los datos recibidos.
5. Informará de su creación de forma exitosa, aumentando la cantidad de vehículos creados en esa sucursal.
6. La página Web solicitará la información del nuevo vehículo creado.
7. El contrato **Vehicle** le pasará los datos que le solicitan.
8. Por último, en la página Web se podrán visualizar todos los vehículos creados en esa sucursal de forma correcta.

Esta ha sido una descripción genérica de los pasos necesarios para la creación de nuevos vehículos. En posteriores apartados, explicaremos de forma más detallada las funciones que han interactuado para realizar esta tarea.

5.4.3. Creación de clientes:

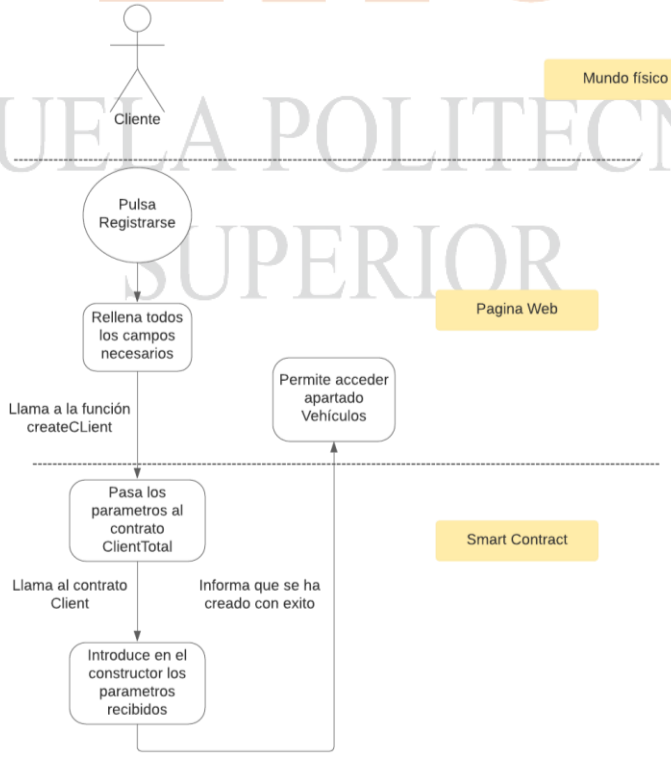

Figura 34: Diagrama de secuencia de creación de clientes

En este diagrama de flujo, se puede ver cómo se desarrolla la interacción para el registro de un nuevo cliente en el sistema.

Los pasos a desarrollar son los siguientes:

1. El cliente decide registrarse para poder acceder a la zona de alquiler de vehículos que ofrece nuestra Web. Por lo tanto, debe pulsar el botón “Registrarse”.
2. A continuación, le aparecerá un desplegable con una serie de campos que tendrá que cumplimentar, de forma correcta, para poder registrarse en el sistema. Los campos son: *Nombre, Apellidos* y *DNI.*
3. Una vez rellenados los campos, pulsará el botón “Registrarse” y se llamará a la función *createCLient*, que se encuentra en el contrato **ClientTotal**.
4. La función *createCLient* pasará los parámetros al constructor del contrato **Client** y este registrará en el sistema el nuevo cliente con todos los datos recibidos.
5. Informará de su creación de forma exitosa y podrá, desde ese momento, acceder desde la cuenta registrada al departamento de vehículos con total operatividad.

5.4.4. Alquiler de vehículos:

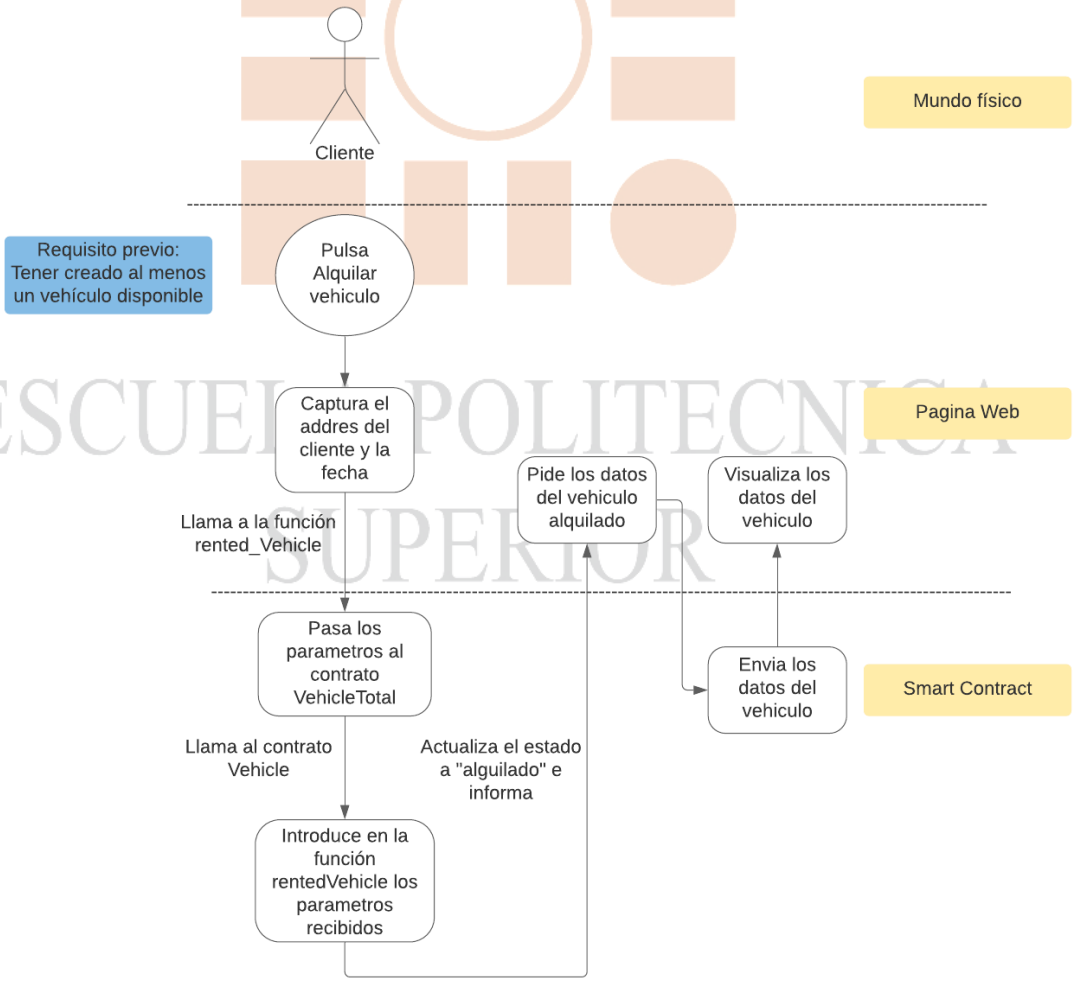

Figura 35: Diagrama de secuencia de alquiler de vehículos

En este diagrama de flujo, se puede apreciar cómo se desarrolla la interacción para la acción de alquilar un vehículo. El único requisito previo es tener al menos un vehículo disponible para poder ser alquilado, de otra manera no se podría alquilar ninguno.

Los pasos a desarrollar son los siguientes:

1. El cliente decide alquilar un vehículo. Por lo tanto, pulsará sobre el botón “Alquilar” del vehículo interesado.
2. La página Web captura la fecha del momento que se pulsó el botón y también captura la dirección de la Wallet desde donde se está haciendo la solicitud de alquilar el vehículo.
3. Todos los parámetros capturados son enviados a la función *rented_Vehicle* del contrato **VehicleTotal**. Este buscará el vehículo seleccionado entre todos los vehículos del sistema.
4. Al encontrar el seleccionado, llamará a la función *rentedVehicle* del contrato **Vehicle**.
5. Se pasarán todos los parámetros para que se pueda actualizar la información del vehículo.
6. Actualizará la información y el vehículo pasará de estar “*Disponible*” a “*Alquilado*”. Por lo tanto, la página Web pedirá la información de este vehículo.
7. El contrato **Vehicle** enviará la información requerida del vehículo en cuestión.
8. Se actualizará nuestra Web, y se visualizarán los vehículos que el cliente tiene actualmente en alquiler.

Esta es una descripción superficial de los pasos necesarios para el alquiler de vehículos. En posteriores apartados, explicaremos de forma más detallada las funciones que han interactuado para realizar esta tarea.

5.4.5. Devolución de vehículos:

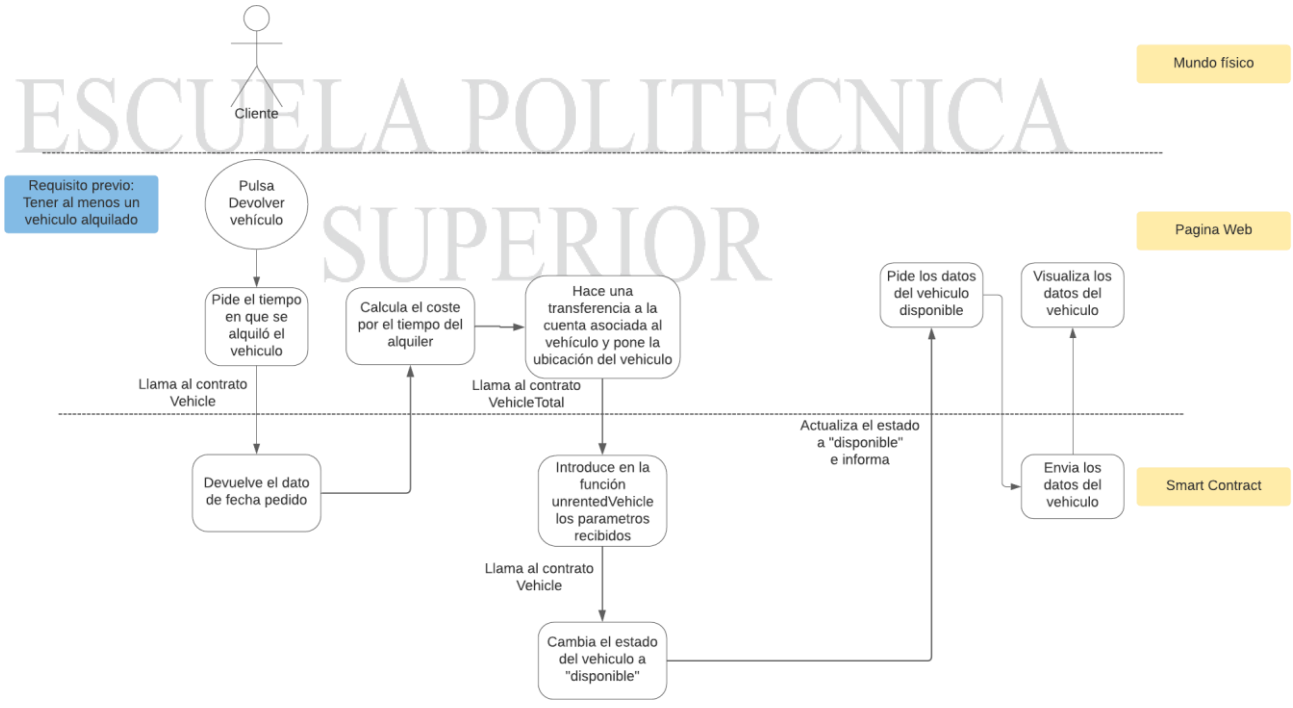

Figura 36: Diagrama de secuencia de devolución de vehículos

En este diagrama de flujo, se puede ver cómo se desarrolla la interacción para la acción de devolución de vehículo. El único requisito previo es tener al menos un vehículo alquilado, porque de otra manera no sería posible la realización de esta acción.

Los pasos a desarrollar son los siguientes:

1. El cliente decide devolver un vehículo. Por lo tanto, pulsará sobre el botón "Devolución" del vehículo interesado.
2. La página Web solicitará al contrato **Vehice** la fecha de alquiler del vehículo.
3. Una vez obtenida la fecha en la que empezó el alquiler, calculará el coste del mismo, teniendo en cuenta el tiempo que ha sido alquilado.
4. El Cliente realizará el pago e indicará la ubicación actual del vehículo.
5. Una vez validada la transferencia de dinero, la página Web llamará a la función *unrented Vehicle* del contrato **VehicleTotal**.
6. La función buscará, entre todos los vehículos del sistema, el vehículo seleccionado a desalquilar.
7. Al encontrarlo, llamará a la función *unrentedVehicle* del contrato **Vehicle**.
8. Pasará todos los parámetros para que se pueda actualizar la información del vehículo.
9. Actualizará la información y el vehículo pasará de estar "*Alquilado*" a "*Disponible*". Por lo tanto, la página Web pedirá la información de este vehículo.
10. El contrato **Vehicle** enviará la información requerida del vehículo en cuestión.
11. La página Web se actualizará y visualizará los vehículos que están disponibles.

Esta es una descripción, a grandes rasgos, de los pasos necesarios para la devolución de vehículos. En posteriores apartados, explicaremos de forma más detallada las funciones que han interactuado para realizar esta tarea.

Una vez explicadas todas las posibles interacciones que se pueden realizar, vamos a abordar de forma más detallada los tres bloques en los que se divide el proyecto. Estos bloques son: Smart Contracts, Interfaz usuario e Interconexión.

CAPÍTULO 6

Smart Contracts

6.1. Introducción.

En el primer bloque, vamos a desarrollar todos los Smart Contracts que han sido necesarios crear para poder realizar el proyecto.

Los contratos necesarios son **siete**. Estos son: dos contratos para la gestión de sucursales, dos contratos para la gestión de vehículos, dos contratos para el almacenamiento de los clientes registrados y un último contrato, obligatorio en cualquier proyecto que se quiera exportar a una red Blockchain, que es el contrato de migraciones.

Empezaremos desarrollando los contratos relacionados con la creación de sucursales. Estos son dos:

6.2. BranchTotal.

El siguiente contrato se utiliza para contener todos los contratos tipo **Branch**, que son los utilizados para la creación de las sucursales.

Variables

Utiliza la siguiente variable:	
Branch[] public branches;	Se utiliza para poder crear todas las sucursales que sean necesarias. Por ello, se utiliza un array dinámico.

Tabla 7: Variables del contrato BranchTotal

La variable "branches" es un array de contratos, tipo **Branch,** que se encuentran agrupados en un array.

Una vez visto la variable que se necesita para el uso de este contrato, vamos a proceder con las funciones que contiene.

Funciones

- function createBranch(…) public

  Esta función nos permite crear las sucursales. Le pasaremos los parámetros capturados en el formulario, que son los siguientes:

  - name: indica el nombre de la sucursal
  - _info: es un número y, dependiendo de su valor, indica un país u otro.
  - _locationName: nombre de la localización de la sucursal
  - _latitudeNum: coordenadas de latitud.
  - _latitudeExp: coordenadas de latitud.
  - _longitudeNum: coordenadas de longitud.
  - _longitudeExp: coordenadas de longitud.
  - _address_to_pay: Wallet asociada a la sucursal. Es utilizada a la hora de realizar el pago de los vehículos.

  Los parámetros se envían al constructor de Branch, que los recibe y crea la sucursal. La sucursal creada se añade al array de branches.

```
Branch newBranch = new Branch(_name, _info, _locationName, _latitudeNum, _latitudeExp, _longitudeNum, _longitudeExp, _address_to_pay);

    branches.push(newBranch);
```

Figura 37: Fragmento de código del contrato BranchTotal

- function getBranchLength() view public returns (uint)

  Esta función nos devuelve la longitud del array de sucursales que están guardados en nuestra Blockchain, siendo el valor más bajo que puede devolver 0.

6.3. Branch.

Este contrato se emplea en la creación de las sucursales, teniendo que rellenar todas las variables que se encuentran a continuación. Entre ellas, podemos destacar el nombre de la sucursal y la localización de esta.

Variables

Utiliza las siguientes variables:	
enum InfoType	Almacena todos los posibles países en los que se puede ubicar la sucursal; es decir, tiene todos los países de la Unión Europea.
InfoType public info;	Variable tipo *InfoType* utilizada para indicar el país donde se encuentra la sucursal.
struct Location	Estructura que almacena las coordenadas del lugar en longitud y latitud.
Location public location;	Variable tipo *Location* utilizada para almacenar las coordenadas de la sucursal.
string public name;	Guarda el nombre de la sucursal.
string public locationName;	Almacena la ubicación de la sucursal. Suele indicarse un nombre de calle.
address payable owner;	Indica la dirección desde la que se ha creado el contrato.
string public address_to_pay;	Se almacena la Wallet que se asigna a la sucursal para poder realizar los pagos a esta.

Tabla 8: Variables del contrato Branch

Una vez vistas las variables utilizadas, vamos a proceder a las funciones.

Función

- constructor(…) public

  El siguiente constructor es la función por defecto utilizada al ejecutarse el contrato. Este recibe todos los parámetros necesarios, que son los mismos que la anterior función denominada *createBranch* del contrato **BranchTotal**.

Guarda todos los datos recibidos en las distintas variables explicadas previamente, siendo distinto el valor del campo "_info" que, dependiendo del valor que tenga, hace que el campo "info" tenga como valor un país u otro.

```
if(_info == 1)
{info = InfoType.Alemania;}
else if(_info == 2)
{info = InfoType.Austria;}
    else if(_info == 3)
{info = InfoType.Belgica;}
    else if(_info == 4)
{info = InfoType.Bulgaria;}
```

Figura 38: Fragmento de código del contrato Branch

Se repite lo que se muestra en la imagen hasta completar los 27 países de los que se compone actualmente la Unión Europea.

Por último, dos consideraciones a tener en cuenta: la primera es que almacenamos en cada posición de la estructura *Location* los parámetros de las coordenadas de latitud y longitud segmentadas y, la segunda, que en el campo *owner* almacenamos la dirección de la persona que crea el contrato.

```
owner = msg.sender;
```

```
location.latNum = _latitudeNum;
location.latExp = _latitudeExp;
location.longNum = _longitudeNum;
location.longExp = _longitudeExp;
```

Figura 39: Fragmento de código del contrato Branch

A continuación, una vez descrito los contratos relacionados con las sucursales, vamos a explicar los contratos relacionados con el almacenamiento de los clientes en el sistema. Proseguiremos explicando estos contratos, debido a su similitud con los comentados anteriormente.

Se componen de dos contratos. Estos son: **ClientTotal** y **Client**. Se desarrollarán seguidamente.

6.4. ClientTotal.

Se utiliza para contener todos los contratos tipo **Client**, que son creados en el momento que un cliente se registre en el sistema.

Variables

Utiliza la siguiente variable:	
Client[] public clientes;	Se utiliza para poder crear y almacenar todos los clientes del sistema. Por ello, se utiliza un array dinámico.

Tabla 9: Variables del contrato ClientTotal

Al igual que en el contrato **BranchTotal**, la variable *clientes* es un array de contratos tipo **Client** que se encuentran agrupados en un array.

Una vez vista la variable que se necesita para el uso de este contrato, vamos a proceder con las funciones que contiene.

Funciones

- function createCLient(…) public:

  Esta función nos permite crear los nuevos clientes a los que pasamos los siguientes parámetros capturados en el formulario:

  -_name: almacena el nombre del usuario.
  -_surname: almacena los apellidos del usuario.
  -_dni: contiene el DNI u otro documento identificativo del usuario.
  -_address: este parámetro, aunque no esté en el formulario, es capturado por la página Web y almacena la Wallet con la que se está registrando.

Los parámetros se envían al construcutor de **Client** que los recibe y crea el del nuevo cliente. El nuevo cliente creado se añade al array de clientes mediante la función push.


```
Client newclient = new Client(_name, _surname, _dni, _address);
    clientes.push(newclient);
```

Figura 40: Fragmento de código del contrato ClientTotal

- function getClientLength() view public returns (uint)

  Esta función nos devuelve la longitud del array de clientes que están guardados en nuestra Blockchain, permitiendo así poder recorrerlo completamente de forma correcta cuando sea necesario.

6.5. Client.

Se emplea para el almacenamiento de los datos de los clientes. Es necesario guardar la información en contratos **Client** para posteriormente poder volcarla en nuestra Web.

Variables

Utiliza las siguientes variables:	
string public name;	Almacena el nombre del cliente.
string public surname;	Almacena los apellidos del cliente.
string public dni;	Guarda el DNI u otro documento identificativo.
address payable owner;	Indica la dirección desde la que se ha creado el contrato.
string public addresss;	Contiene la dirección de la Wallet asociada al cliente.

Tabla 10: Variables del contrato Client

Una vez vistas las variables utilizadas, vamos a proceder a las funciones.

Función

- constructor(…) public

El siguiente constructor es la función por defecto utilizada al ejecutarse el contrato. Este recibe todos los parámetros necesarios, que son los mismos que la anterior función denominada *createCLient* del contrato ***clientTotal***.

Los campos se almacenan como se muestran a continuación, siendo el campo "owner" distinto al resto. Del mismo modo que en el contrato **Branch**, almacenamos directamente la dirección de la persona que genera el contrato, en este caso es la misma que se almacena en "addresss", obteniendo así una doble seguridad.

```
 name=_name;
 surname=_surname;
dni=_dni;
addresss=_address;
owner = msg.sender;
```

Figura 41: Fragmento de código del contrato Client

Recordaremos que la función "msg.sender" contiene la dirección de la Wallet desde la que se está generando o llamando al contrato en cuestión. En este caso el del futuro cliente.

Por último, solo nos resta abordar los contratos relacionados con los vehículos. Estos contratos se explican los últimos, debido a la complejidad que manejan respecto a los comentados anteriormente. Estos contratos se basan en dos Smart Contracts: **VehicleTotal** y **Vehicle**.

6.6. VehicleTotal.

Se utiliza para contener todos los contratos tipo **Vehicle,** que son creados en el momento que un administrador registre un nuevo vehículo en una sucursal. Este contrato también tiene las funciones de "Alquilar" y "Devolver" un vehículo en cuestión.

Variables

Utiliza la siguiente variable:	
Vehicle[] public vehicles;	Se utiliza para almacenar los distintos vehículos creados por las sucursales. Por ello, se utiliza un array dinámico.

Tabla 11: Variables del contrato VehicleTotal

La variable "vehicles" es un array de contratos tipo **Vehicle** que se encuentran agrupados en un array.

Una vez vista la variable que se necesita para el uso de este contrato, vamos a ver las funciones que contiene.

Funciones

- function createVehicle(…) public

  Esta función nos permite crear los vehículos. Le pasaremos los parámetros capturados en el formulario que se encuentra dentro de cada sucursal. Estos son los siguientes:

- _locationName: nombre de la localización del vehículo.
- _latitudeNum: coordenadas de latitud.
- _latitudeExp: coordenadas de latitud.
- _longitudeNum: coordenadas de longitud.
- _longitudeExp: coordenadas de longitud.
- vehicle_registration: matrícula del vehículo.
- _VehicleType: tipo de vehículo. Es un valor numérico, cuyo rango es [1,9], que indica el tipo de vehículo.
- _rented: este valor no es capturado desde el formulario. Es el valor, por defecto, utilizado para crear un vehículo y lo ponemos a "*True*"; es decir, a disponible.

- _branch: este valor tampoco se captura del formulario, pero indica el nombre de la sucursal en la que ha sido creada el vehículo, para poder enlazarlo a esta.

Los parámetros se envíarán al constructor de **Vehicle** que los recibe y crea el nuevo vehículo. Este se añade al array de vehicles.

```
Vehicle newVehicle = new Vehicle( locationName, latitudeNum, latitudeExp, longitudeNum, longitudeExp, vehicle_registration,_VehicleType, _rented, _branch);

    vehicles.push(newVehicle);
```

Figura 42: Fragmento de código del contrato VehicleTotal

➢ function getVehicleLength() view public returns (uint)

Esta función nos devuelve la longitud del array de vehículos que están guardados en nuestra Blockchain.

➢ function rented_vehicle(…)public

Esta función permite el alquiler de vehículos, recibiendo la dirección del vehículo a alquilar, la fecha y la dirección de la Wallet del cliente.

Se recorrerá el array de "vehicles" y se buscará el *vehicle* que tenga la misma dirección del que deseamos alquilar. Una vez encontrado, se llamará a una función del contrato **Vehicle**, denominada *rentedVehicle,* y le pasaremos los parámetros de la fecha y la dirección de la Wallet del cliente.

```
Vehicle Vehicle_to_rented = Vehicle(_vehicle);

for (uint i = 0; i < vehicles.length; i++){

    if(vehicles[i] == Vehicle_to_rented){
        vehicles[i].rentedVehicle(_direccion, _day, _month, _year);
    }
}
```

Figura 43: Fragmento de código del contrato VehicleTotal

➢ function unrented_vehicle(…)public

Esta función permite la devolución de vehículos, recibiendo del vehículo a devolver las coordenadas de la dirección, y el nombre de la localización donde se encuentra.

Se recorrerá el array de "vehicles" y se buscará el *vehicle* que tenga la misma dirección que el vehículo que deseamos devolver. Una vez encontrado, se llamará a la función del contrato **Vehicle** *unrentedVehicle* y se le pasarán los parámetros de las coordenadas de latitud y longitud. También se le pasará el nombre de la localización para mayor exactitud.

```
Vehicle Vehicle_to_unrented = Vehicle(_vehicle);

for (uint i = 0; i < vehicles.length; i++){

    if(vehicles[i] == Vehicle_to_unrented){
        vehicles[i].unrentedVehicle(latitudeNum, latitudeExp, longitudeNum, longitudeExp, locationName);
    }
}
```

Figura 44: Fragmento de código del contrato VehicleTotal

6.7. Vehicle.

Se emplea para el almacenamiento de los datos de los vehículos registrados en las sucursales. Este contrato tiene la capacidad de almacenar los datos del vehículo, e implementa las funciones de "Alquilar" y "Devolver" el vehículo.

Variables

Utiliza las siguientes variables:	
enum VehicleType	Almacena todos los posibles tipos de vehículos que se pueden alquilar. Estos son: económico, medio, ejecutivo, lujo, monovolumen, minibús, furgoneta y motocicleta.
VehicleType public vehicletype	Variable tipo *VehicleType* utilizada para indicar el tipo de vehículo que es.
struct Location	Estructura que almacena las coordenadas del lugar en longitud y latitud.
Location public location	Variable tipo *Location* utilizada para almacenar las coordenadas del vehículo.
string public locationName;	Almacena la ubicación del vehículo. Suele indicarse un nombre de calle.
string public vehicle_registration	Indica la matrícula del vehículo.
address payable owner	Indica la dirección desde la que se ha creado el contrato.
bool public rented	Indica, por medio de una variable booleana, si el vehículo está disponible o se encuentra actualmente alquilado, siendo "True" Disponible y siendo "False" Alquilado.
string public direccion	Almacena el *address* de la cuenta que alquiló el vehículo, teniendo un valor "0x0000000000000000000000000000000000000000" cuando el vehículo está disponible; es decir, no está alquilado por ningún cliente.

string public branch	Guarda el nombre de la sucursal en la que está creado el vehículo para poder enlazarlo con la sucursal.
uint public day	Guarda el día en el que fue alquilado el vehículo. Cuando no está alquilado, tiene un valor 0 por defecto.
uint public month	Guarda el mes en el que fue alquilado el vehículo. Cuando no está alquilado, tiene un valor 0 por defecto.
uint public year	Guarda el año en el que fue alquilado el vehículo. Cuando no está alquilado, tiene un valor 0 por defecto.

Tabla 12: Variables del contrato Vehicle

Una vez vistas las variables utilizadas, vamos a proceder a las funciones.

Funciones

- constructor(…) public

El siguiente constructor es la función por defecto al ejecutarse el contrato. Este recibe todos los parámetros necesarios para poder almacenarlos en las variables explicadas anteriormente.

Guarda en las distintas variables los valores recibidos por la función *createVehicle* y, como en el caso de *createBrach*, la variable que recibe el constructor denominada "_VehicleType" indica el tipo de vehículo que se refiere. La variable puede tener valores entre uno y nueve inclusive, como se muestra a continuación en la siguiente imagen.

```
if(_VehicleType == 1)
   {
vehicletype =   VehicleType.Economico;
   }
   else if(_VehicleType == 2)
   {
vehicletype =   VehicleType.Medio;
   }
   else if(_VehicleType == 3)
   {
vehicletype =   VehicleType.Ejecutivo;
   }
   else if(_VehicleType == 4)
   {
vehicletype =   VehicleType.Lujo;
   }
```

Figura 45: Fragmento de código del contrato Vehicle

Se sigue la misma estructura hasta completar los ocho tipos de vehículos que tenemos registrados en el sistema.

Por último, tenemos la variable "owner" que registra la dirección de la persona que realiza el contrato; es decir, la creación del vehículo. Cabe destacar que, por defecto, tenemos las variables "day", "month", "year" y "dirección" con valor de 0, debido a que cuando se crea el vehículo todavía no ha sido alquilado en ninguna ocasión y, por ende, le ponemos el valor 0 para que quede registrado.

```
owner = msg.sender;
direccion="0x0000000000000000000000000000000000000000";
branch = _branch;
day = 0;
month = 0;
year = 0;
```

Figura 46: Fragmento de código del contrato Vehicle

- function rentedVehicle(…)public

Esta función nos permite registrar el alquiler del vehículo, modificando las variables del contrato que conciernen a la fecha, situación actual del vehículo y persona que se dispone a alquilarlo.

En cuanto a la fecha, se introduce la fecha del sistema capturada por la página Web, debido a que en el momento de la devolución se utilizará esta fecha para poder calcular el coste de alquiler.

En relación a otras variables que se modifican, se introduce en la variable "dirección" el address de la Wallet del cliente que realiza la operación en modo string. En "owner" se guarda el mismo dato que en "dirección", pero esta vez se guarda en tipo address para mayor seguridad. Por último, se modifica la variable "rented" a false, indicando que el vehículo se encuentra actualmente en alquiler.


```
owner = msg.sender;
rented = false;
direccion = aux;
day = _day;
month = _month;
year = _year;
```

Figura 47: Fragmento de código del contrato Vehicle

- function unrentedVehicle(…)public

Esta función nos permite registrar la devolución del vehículo, modificando las variables del contrato que se refieren a las coordenadas de su ubicación, nombre de localización o de la calle, dirección y situación del vehículo.

Modificamos las coordenadas como en el caso del constructor; es decir, guardamos en la estructura creada la longitud y la latitud nuevas del vehículo. En el caso del nombre de la localización, se guarda la nueva que el cliente previamente ha tenido que introducir para poder devolverlo.

En lo que respecta a las variables de “dirección”, “owner” y “rented”, a estas dos primeras se les introduce el valor de:

0x0000000000000000000000000000000000000000

en forma de string y en forma de address para indicar que el vehículo ha sido devuelto y que ya no pertenece al cliente. En lo referente a “rented”, se cambia el valor a true, indicando que vuelve a estar disponible.

Todo lo descrito, se puede observar en la siguiente imagen.

```
    owner = 0x0000000000000000000000000000000000000000;
    direccion = "0x0000000000000000000000000000000000000000";

locationName = _locationName;

location.latNum = latitudeNum;
location.latExp = latitudeExp;
location.longNum = longitudeNum;
location.longExp = longitudeExp;
    rented = true;
```

Figura 48: Fragmento de código del contrato Vehicle

6.8. Migrations.

Este contrato viene por defecto en *truffle box*. Se compone de todo lo necesario para poder migrar el contrato de forma correcta hacia nuestra Blockchain. Como este contrato no ha sido generado por nosotros, procederemos simplemente a enumerar las funciones y las variables que utiliza y, por último, mostraremos una imagen con la totalidad del contrato Migrations.sol.

Variables

Utiliza las siguientes variables:	
uint public last_completed_migration	Almacena un número que corresponde al último script de "migración" aplicado, que se encuentra en la carpeta de migraciones. Empieza siempre por el valor 1 y va aumentando. En nuestro caso va a llegar hasta el valor 2, debido a que tenemos dos scripts para poder migrar todos los contratos.
address public owner	Indica la dirección desde la que se ha creado el contrato.

Tabla 13: Variables del contrato Migrations

Funciones

- constructor(…) public

El siguiente constructor es la función, por defecto, al ejecutarse el contrato.

- modifier restricted()
- function setCompleted(…) public restricted
- function upgrade(…) public restricted

Por último, se muestra una imagen del contrato **Migrations.sol** completo para que se pueda apreciar todo el contenido, debido a que no vamos a profundizar en él.

```
pragma solidity >=0.4.21 <0.6.0;

contract Migrations {
  address public owner;
  uint public last_completed_migration;

  constructor() public {
    owner = msg.sender;
  }

  modifier restricted() {
    if (msg.sender == owner) _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }

  function upgrade(address new_address) public restricted {
    Migrations upgraded = Migrations(new_address);
    upgraded.setCompleted(last_completed_migration);
  }
}
```

Figura 49: Código completo del contrato Migrations

CAPÍTULO 7

Front-End

7.1. Otras configuraciones.

Antes de explicar el *Front-End*, es necesario abordar otras configuraciones para poder migrar los contratos a la red Blockchain y después poder interactuar con estos.

Primero, vamos a mostrar una imagen de cómo están estructurados los contratos descritos en el anterior capítulo, y los archivos de la carpeta *migrations* que vamos a desarrollar a continuación.

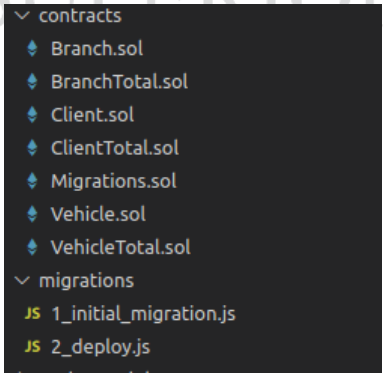

Figura 50.1: Estructura de los smart contracts y archivos de migración

Este apartado no es importante a la hora de desarrollar los contratos o el *Front-End,* pero es esencial para su correcto funcionamiento y una correcta interacción entre ambos. Por ello, vamos a explicar los siguientes archivos:

7.1.1 initial_migration.js

Este archivo se encuentra dentro de la carpeta *migrations* y su función es únicamente migrar el contrato **Migrations.sol**, explicado en el apartado de Smart Contracts.

```
var Migrations = artifacts.require("./Migrations.sol");

module.exports = function(deployer) {
  deployer.deploy(Migrations);

};
```

Figura 50.2: Fragmento de código del archivo initial_migration

Como se puede apreciar, necesita el archivo **Migrations.sol** para poder migrarlo a nuestra red.

7.1.2 deploy.js

Este archivo, al igual que el anterior, se encuentra en la carpeta de *migrations*. Tienen un número delante del nombre que indica el número de script, siendo distinto para cada **.js* de la carpeta. Este archivo contiene las sentencias necesarias para poder migrar los Smart Contracts explicados anteriormente, que son: **VehicleTotal**, **ClientTotal** y **BranchTotal**.

```
var VehicleTotal = artifacts.require("VehicleTotal");
var BranchTotal = artifacts.require("BranchTotal");
var ClientTotal = artifacts.require("ClientTotal");

module.exports = function(deployer) {
  deployer.deploy(VehicleTotal);
  deployer.deploy(BranchTotal);
  deployer.deploy(ClientTotal);
};
```

Figura 51: Código completo del archivo deploy

Como en el caso anterior, necesitará los archivos **VehicleTotal**, **BranchTotal** y **ClientTotal** para poder desplegarlos en la red Blockchain.

7.1.3 truffle-config.js

Este archivo es necesario para saber a qué tipo de red vamos a migrar los contratos y poder indicar dónde se encuentra la red para poder desplegarlos.

En este archivo ha sido necesario modificar únicamente unos apartados. Las actuaciones han consistido en activar una serie de líneas, que por defecto están desactivadas, y la modificación de estas.

```
module.exports = {

  contracts_build_directory: path.join(__dirname, "./src/contracts"),
  networks: {

    development: {
     host: "192.168.1.115",     // Localhost (default: none)
     port: 7545,            // Standard Ethereum port (default: none)
     network_id: "*",       // Any network (default: none)
    }
```

Figura 52: Fragmento de código del archivo truffle-config

Ha sido necesario añadir, dentro de *module.exports,* la línea correspondiente a **contract_build_directory** que indica dónde se guardarán los archivos .json que se creen una vez desplegados los contratos.

También se ha modificado la dirección donde se despliegan los contratos. Por defecto, se encuentra en la dirección 127.0.0.1 puerto 7545 y ha sido modificada, en nuestro caso, a 192.168.115 puerto 7545 para poder desplegarlos en Ganache, que se aloja en esa dirección. Esto se ha realizado para poder acceder desde cualquier dispositivo en red, sin tener que encontrarse en la misma máquina que donde se encuentra corriendo Ganache.

7.2. Front-End.

En este punto nos vamos a centrar en el *Front-End*; es decir, en la parte visual del portal Web.

Antes de comenzar a introducir los componentes creados y cómo se relacionan entre ellos, hay que mencionar los paquetes que han sido necesarios instalar por medio de NPM para el correcto funcionamiento de la aplicación que se ha desarrollado:

- @drizzle/React-components
- @drizzle/React-plugin
- @drizzle/store
- bootstrap
- drizzle
- Drizzle-react
- jqery
- Qrcode.react
- react
- React-bootstrap-carousel
- React-dom
- React-scripts
- reactstrap
- serve
- Web3

Cabe destacar que la mayoría de los paquetes vienen incluidos con el despliegue de una "Truffle Box", permitiéndonos obtener en un corto periodo de tiempo, los esenciales que nos permitan poder trabajar con React.

Vamos a empezar a explicar este punto del proyecto, gracias a un esquema en el que se muestran todos los componentes utilizados para el desarrollo de este apartado.

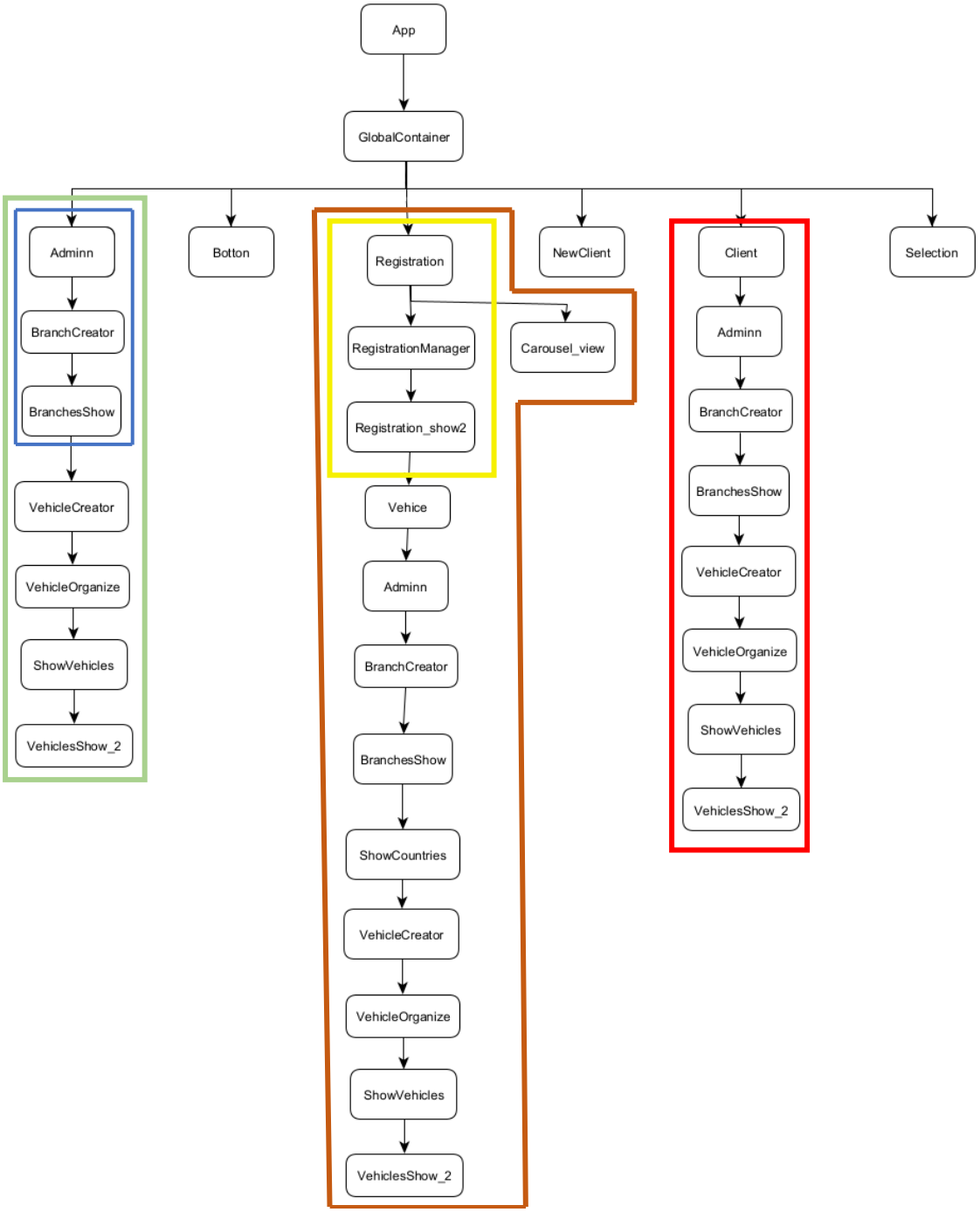

Figura 53: Esquema de la estructura de la página Web

Como se puede observar en el esquema, el *Front-End* se divide en **seis** secciones. Todas son necesarias para que el portal Web funcione correctamente.

En el esquema anterior se han señalizado con colores cada una de estas secciones. Con ello se persigue una mejor identificación de los componentes utilizados en la realización de acciones, que pasaremos a explicar a continuación.

7.2.1 Descripción del bloque de color azul.

En este color están representados los componentes que son necesarios para crear nuevas sucursales y, después, poder mostrar en una lista todas las creadas hasta ese momento.

Esto se realizará gracias al componente "**Adminn"** (crear las sucursales) y los componentes "**BranchCreator**" y "**BranchesShow**", que se utilizan para obtener las sucursales de la Blockchain y mostrarlos en la página Web.

7.2.2 Descripción del bloque de color verde.

En este color están englobados los componentes necesarios para que cada sucursal pueda crear vehículos y se puedan visualizar, tanto la información de la localización del vehículo valiéndonos de un QR, como el resto de la información de forma más extensa.

Los componentes del color azul permiten visualizar todas las sucursales creadas. El componente "**VehicleCreator**" para crear todos los vehículos, y los componentes "**VehicleOrganize**", "**ShowVehicles**" y "**VehiclesShow_2**" para obtener los vehículos de la red Blockchain y mostrarlos en la página Web.

7.2.3 Descripción del bloque de color Amarillo.

Este color contiene los componentes que se necesitan para obtener, desde nuestra red, los clientes ya registrados y guardados, y poder mostrarlos en la página Web.

Se utilizan los componentes "**Registration**" y "**RegistrationManager**" para obtener de la red Blockchain los clientes, y "**registration_show2**" para mostrarlos en la página Web a través de un QR, o mediante la lista de toda la información.

7.2.4 Descripción del bloque de color Marrón.

En este color se incluyen los componentes para que los clientes, una vez registrados, puedan acceder a los vehículos adscritos a las sucursales y se pueda realizar su alquiler.

Esto se consigue gracias a los componentes utilizados en la zona remarcada en color amarillo a fin de averiguar si el cliente está registrado y, si es así, se utilizan el resto de los componentes para mostrar las sucursales y, dentro de estas, visualizar los vehículos que contienen utilizando la misma estructura que en el color verde. La única diferencia en este color respecto al verde, es que se permite alquilar los vehículos y en el verde no. Esto es debido a que el color verde está pensado para el uso de los administradores, no de los clientes.

Por último, hay que destacar que el componente "**Carousel_view**" se utiliza para mostrar las imágenes de las distintas modalidades de vehículos en modo Carrusel, para que los clientes puedan apreciar mejor los distintos tipos existentes.

7.2.5 Descripción del bloque de color rojo.

En este último color se engloban los componentes que interaccionan en la visualización de los clientes que tienen en ese momento vehículos en alquiler.

Esto se consigue gracias al componente "**Client**", que nos permite introducir un parámetro que únicamente nos posibilitará visualizar los vehículos que estén alquilados y que, a su vez, se encuentren en la cuenta que en ese momento está operativa en la página Web. Al igual que en el color verde, la misión del resto de los componentes es la de visualizar los vehículos con la única diferencia que, en este caso, tienen que haber sido alquilados previamente por otros clientes.

Hay otras interacciones que se realizan en la página Web pero, pero debido a que solo interviene un componente, se abordará más adelante.

A continuación, vamos a explicar la misión de cada componente creado y, por último vamos a utilizar un esquema para poder representar, de forma visual, los componentes que se utilizan para acceder a cada zona de la página Web.

Antes de profundizar en los componentes, hay que explicar dos archivos fundamentales que se denominan ***drizzle.js*** e ***index.js***.

7.2.6 Drizzle.js

Este archivo sirve para indicar dónde se debe ejecutar el *Front-End* y nos permite definir qué contratos son los que se pueden utilizar a la hora de interactuar con la página Web. También puntualizamos las especificaciones que utilizamos de Drizzle. Todo esto se puede observar en la siguiente imagen.

```
1  import { Drizzle, generateStore } from "drizzle";
2
3  import VehicleTotal from './contracts/VehicleTotal.json';
4  import BranchTotal from './contracts/BranchTotal.json';
5  import ClientTotal from './contracts/ClientTotal.json';
6
7  const options = {
8
9      contracts: [VehicleTotal, BranchTotal, ClientTotal],
10     web3: {
11         fallback: {
12             type: "ws",
13             url: "ws://127.0.0.1:7545"
14         }
15     }
16 };
17
18 const drizzleStore = generateStore(options);
19
20 const drizzle = new Drizzle(options, drizzleStore);
21
22 export default drizzle;
```

Figura 54: Código completo del archivo Drizzle

Se puede comprobar, como se ha mencionado anteriormente, la indicación de qué contratos vamos a utilizar. En este caso: **VehicleTotal**, **BranchTotal** y **ClientTotal** y, previamente, dónde se encuentran para poder acceder a sus campos. También se puede ver dónde se despliega el portal web, siendo la dirección 127.0.0.1:7475. Cuando se quiera acceder a él desde un equipo externo, es necesario introducir la dirección de pública de la máquina donde esta se aloja.

7.2.7 Index.js

Index.js es un fichero esencial para el funcionamiento de todo lo que se explicará posteriormente, debido a que en este documento se indican las librerías que se van a utilizar para el desarrollo de los componentes.

7.3 Funcionamiento de los componentes

7.3.1 App.js

Este componente es el primero que se carga. Sirve para comprobar que el cliente web que intente acceder a la Web cumpla con todos los requisitos necesarios. Lo fundamental es tener una Wallet asociada al navegador para poder realizar transacciones a través de esta. Este requisito se puede cumplir de forma sencilla al instalar la extensión MetaMask, explicada en el apartado Tecnologías Utilizadas.

```
if (!initialized) return "Cargando...";

return (
  <div className="App">
    <GlobalContainer  drizzle={drizzle} drizzleState={drizzleState}/>
  </div>
  );
```

Figura 55: Fragmento del componente App

Como se ve en la imagen superior, si no está *initialized* mostrará por pantalla únicamente el mensaje "Cargando…". En cambio, si se cumple la condición de MetaMask, es decir, está *initialized*, llamará al componente "**GlobalContainer"** que contiene la totalidad de la página Web.

7.3.2 GlobalContainer

Este componente es esencial para poder cambiar entre las distintas páginas del sitio Web. Tiene un estado denominado "showView" que nos permite modificar su valor entre "*admin*", "*vehicle*", "*client*", "*NEWclient*" y "*Registration*". Estos valores nos permiten mostrar una u otra página Web, dependiendo de qué valor tenga en ese momento. Modificamos los valores por medio de la siguiente función:

```
changeViewNEWClient = async e =>{
  if(e) e.preventDefault();
  const accounts = await window.ethereum.enable();
  const account = accounts[0];
  const showView = "NEWclient";
  this.setState({showView, account});

}
```

Figura 56: Fragmento del componente GlobalContainer

Una vez llamada a la función *changeViewNEWClient*, cambiará el estado de "showView" a "*NEWclient*" y, a la vez, capturará la cuenta que está en ese momento activa a la hora de realizar el cambio del estado. Esto último es necesario para poder utilizar la información del *address* de la cuenta para otros componentes. Por ejemplo, se utiliza a la hora de registrar un nuevo cliente. Se captura la cuenta que se está utilizando para registrarse y así queda asociada al cliente.

Hay tantas funciones como estados posibles haya en el sistema. En nuestro caso hay un total de seis funciones similares.

En el componente también se captura la hora del sistema a fin de utilizarla en el momento en el que se decida alquilar algún vehículo, o se decida devolverlo. Esta información es enviada a los componentes "**Adminn**", "**Registration**", "**NEWClient**" o "**Client**", dependiendo del valor del estado "showView".

Por último, también llama al componente "**Selection**", que es el encargado de modificar el valor del estado dependiendo de la opción seleccionada por el usuario.

7.3.3 Adminn.

Este componente se utiliza para crear las sucursales y visualizarlas, o dependiendo desde donde se acceda, únicamente visualiza las sucursales.

Para crear las sucursales se utilizará un formulario donde se tendrán que cumplimentar todos los campos necesarios para el contrato, como se muestra a continuación.

```
<Form onSubmit={this.createBranch.bind(this)} className="formBox">
  <FormGroup>
    <Label>Nombre </Label>
    <Input type="text" id="branchName" placeholder="Nombre de la sucursal" required />
  </FormGroup>
  <FormGroup>
    <Label>Localización </Label>
    <Input type="text" id="branchNameLocation" placeholder="Localización del lugar" required />
  </FormGroup>
  <FormGroup>
    <Label>Latitud</Label>
    <Input type="text" id="branchLatitude" placeholder="Latitud" required />
  </FormGroup>
  <FormGroup>
    <Label>Longitud</Label>
    <Input type="text" id="branchLongitude" placeholder="Longitud" required />
  </FormGroup>
  <FormGroup>
  <FormGroup>
    <Label>Cuenta de la empresa </Label>
    <Input type="text" id="branchaddress_to_pay" placeholder="Wallet de la empresa" required />
  </FormGroup>

    <FormGroup>
                                <Label>Pais de la sucursal</Label>
                                <Input type="select" id="branchInfo" required>
                                  <option value={1}> Alemania </option>
                                  <option value={2}> Austria </option>
                                  <option value={3}> Bélgica </option>
                                  <option value={4}> Bulgaria </option>
                                  <option value={5}> Chequia </option>
                                  <option value={6}> Chipre </option>
                                  <option value={7}> Croacia </option>
                                  <option value={8}> Dinamarca </option>
                                  <option value={9}> Eslovaquia </option>
                                  <option value={10}> Eslovenia </option>
                                  <option value={11}> España </option>
                                  <option value={12}> Estonia </option>
                                  <option value={13}> Finlandia </option>
                                  <option value={14}> Francia </option>
                                  <option value={15}> Grecia </option>
                                  <option value={16}> Hungría </option>
                                  <option value={17}> Irlanda </option>
                                  <option value={18}> Italia </option>
                                  <option value={19}> Letonia </option>
                                  <option value={20}> Lituania </option>
                                  <option value={21}> Luxemburgo </option>
                                  <option value={22}> Malta </option>
                                  <option value={23}> Países Bajos </option>
                                  <option value={24}> Polonia </option>
                                  <option value={25}> Portugal </option>
                                  <option value={26}> Rumanía </option>
                                  <option value={27}> Suecia </option>
                                </Input>
                    </FormGroup>


  </FormGroup>
  <Button type="submit">Crear nueva sucursal</Button>
</Form>
```

Figura 57: Fragmento del componente Adminn

Como se puede observar, se rellenarán los campos de: “Nombre de la sucursal”, “Localización del lugar”, “Latitud”, “Longitud” y “Wallet de la empresa”. Una vez cumplimentados, se guardarán en sus variables y serán capturados por la función *createBranch,* cuya misión será la de obtener los campos del formulario y llamar al contrato **BranchTotal** que crea las sucursales.

```
const createBranchId = contract.methods["createBranch"].cacheSend( branchName, branchInfo, branchNameLocation,branchLatitude, branchLatitudeExp, branchLongitude, branchLongitudeExp,branch_address_to_pay, {
  from: drizzleState.accounts[0], gas: 4712388,
  gasPrice: 10000000000
```

Figura 58: Fragmento del componente Admin

Como se puede apreciar en la imagen, es una parte de la función *createBranch* perteneciente al contrato **BranchTotal,** el cual se encargará de capturar todos los datos.

7.3.4 BranchCreator

Este componente lo utilizamos para obtener de la red Blockchain las sucursales que se encuentran creadas por medio de la función *map*.

```
var components = myBranches.map((branch, index) => {
    if (((branch && branch.value) !== undefined)){

    if(this.props.viewClient && w === 0)
       {

        if(this.props.adminselect)
        {
           w = w + 1;
      return (
                 <div>


                    <BranchesShow
                     account={this.props.account}
                      show_rented={this.props.show_rented}
                       adminselect={this.props.adminselect}
                        key={index}
                        vehicleselect={this.props.vehicleselect}
                        dia={this.props.dia}
                        mes={this.props.mes}
                        anyo={this.props.anyo}
                        viewClient={this.props.viewClient}
                         address={branch.value}
                         client_view_pay={this.props.client_view_pay}
                         branch_to_pay={this.props.branch_to_pay}
                         countrie_number={this.props.countrie_number}
                          index={index}
                         branch_create_view={this.props.branch_create_view}
                           drizzle={this.props.drizzle}
                            drizzleState={this.props.drizzleState}
                            /></div>);
```

Figura 59: Fragmento del componente BranchCreator

Conseguiremos obtener todas las sucursales que se encuentran guardadas en nuestra red y, posteriormente, por cada sucursal se llamará al componente "**BranchesShow**" para así continuar con el proceso de mostrar la información a través de la Web. Se llamará a tantos componentes como sucursales existan.

7.3.5 BranchesShow

La finalidad de este componente es mostrar la información de las sucursales de forma visual y, dependiendo desde donde se acceda a este componente, nos permitirá seguir con el proceso de mostrar los vehículos mediante la llamada a otros componentes.

```
<Media left href="#">
  <Media object src="img/branch/sucursal.png" alt="" className="pointer rounded-circle"/>
  </Media>
  <Media heading onClick={this.props.showcreateVehicle} className="pointer subrayar">
 {"Sucursal "+(Name)}
 </Media>
    <Media body className="marginLeft">
        <Row>
                                <Col>
        <ul>
         <li>Se sitúa en: {locationName}.  <a target="_blank" rel="noopener noreferrer" href={"https://www.google.com/maps/search/?api=1&query="+String(latitude)+","+String(longitude)}>
          <li>Nombre: {Name}</li>
         </ul>
         </Col>
                        <Col>
                            <iframe name="gMap" width="100%" height="200" frameborder="0" scrolling="no" marginheight="0" marginwidth="0" src={`https://www.google.com/maps/embed/v1/place?
                        </Col>
                        </Row>
```

Figura 60: Fragmento del componente BranchesShow

Dependiendo desde donde se acceda, de forma visual se mostrarán por medio de una lista las sucursales, en la que se podrán apreciar los parámetros más importantes. En este caso se muestra la localización por medio de una URL que nos llevará a la localización en Google Maps, el nombre de la sucursal, y también aparecerá un *iframe* que nos mostrará, en un pequeño cuadrado en la derecha, la posición de la sucursal sin necesidad de hacer clic en la URL.

En el caso de que se acceda por otra vía que no sea la mostrada en este caso, además de mostrar la sucursal con información adicional como la dirección de la Wallet que se utiliza para realizar los pagos, también llamará a un componente denominado "**VehicleCreator**", que se encargará del proceso de crear y mostrar vehículos o simplemente mostrar.

Por último, también se puede acceder de otra forma en la que se llamará a un componente cuyo nombre es "**ShowCountries**", que nos permitiría mostrar la información de la sucursal y la de los vehículos, pero solamente aparecerían las sucursales del país que haya sido seleccionado.

Cabe destacar que, para evitar que aparezca toda la información por pantalla a la vez, se utiliza una función que nos permite ocultar o mostrar la información a nuestro gusto. Por defecto la información permanece oculta y solo se muestra lo esencial, pero cuando se hace clic sobre la información que nos interesa, esta se despliega gracias a la función *showcreatevehicle*.

```
  showcreateVehicle(e){
if(e) e.preventDefault();
let showcreateVehicle;
if (this.state.showcreateVehicle) {
  showcreateVehicle=false;
    this.setState( {showcreateVehicle});
}else{
  showcreateVehicle=true;
    this.setState( {showcreateVehicle});
}
}
```

Figura 61: Fragmento del componente BranchesShow

7.3.6 ShowCountries

La utilización de este componente es, al igual que el antes mencionado "**BranchesShow",** mostrar la información de las sucursales de forma visual y llamar al componente "**VehicleCreator**" para continuar con el proceso de mostrar los vehículos.

```
<Media left href="#">
    <Media object src="img/branch/sucursal.png" alt="" onClick={this.props.showcreateVehicle} className="pointer rounded-circle"/>
    </Media>
  <Media body className="marginLeft">
    <Media heading onClick={this.props.showcreateVehicle} className="pointer subrayar">
      {"Sucursal "+(this.props.Name)}
      </Media>
      <Row>
                                <Col>
      <ul>
      <li>Se sitúa en: {this.props.locationName}.  <a target="_blank" rel="noopener noreferrer" href={"https://www.google.com/maps/search/?api=1&query="+String(this.props.latitude)+","
        <li>Nombre: {this.props.Name}</li>
        <li>Cuenta de la sucursal: {this.props.branch_topay}</li>
        </ul>
        </Col>
                            <Col>
                                <iframe name="gMap" width="60%" height="200" frameborder="0" scrolling="no" marginheight="0" marginwidth="0" src={`https://www.google.com/maps/embed/v1/
                            </Col>
                            </Row>
```

Figura 62: Fragmento del componente ShowCountries

Como se puede observar, su estructura es muy similar al del componente "**BranchesShow**", que ya se explicó en el apartado anterior.

Ha sido necesario crear este componente debido a que, cuando se muestran los países en los cuales se encuentran nuestras sucursales, se ha tenido que crear un nivel nuevo de profundidad a fin de poder ahorrar código y mostrar únicamente las sucursales del país seleccionado.

Una vez mostrada la sucursal, se llamará al componente "**VehicleCreator**" para que interactúe con los vehículos.

7.3.7 VehicleCreator

Este componente se utiliza para crear y llamar a otros componentes que nos muestran los vehículos, o al ser invocado desde otra ubicación, solamente iniciará el proceso de visualización de estos.

```
else if(this.props.show_statecreateVehicle)//estamos en modo crear vehiculos
{

            return (
              <Container fluid>
                <h3 className="margins">Se han creado {(vehicleLength && vehicleLength.value) || 0} vehiculos.</h3>
                <Form onSubmit={this.createVehicle.bind(this)} className="formBox">
                  <FormGroup>
                    <Label>Nombre </Label>
                    <Input type="text" id="vehicleName" placeholder="Matricula del Vehiculo" required />
                  </FormGroup>
                  <FormGroup>
                    <Label>Localización </Label>
                    <Input type="text" id="vehicleNameLocation" placeholder="Nombre de la localización" required />
                  </FormGroup>
                  <FormGroup>
                    <Label>Latitud</Label>
                    <Input type="text" id="vehicleLatitude" placeholder="Latitud" required />
                  </FormGroup>
                  <FormGroup>
                    <Label>Longitud</Label>
                    <Input type="text" id="vehicleLongitude" placeholder="Longitud" required />
                  </FormGroup>
                  <FormGroup>
                                <Label>Tipo de producto</Label>
                                <Input type="select" id="vehicleType" required>
                                  <option value={2}> Economico </option>
                                  <option value={3}> Medio </option>
                                  <option value={4}> Ejecutivo </option>
                                  <option value={5}> Lujo </option>
                                  <option value={6}> Monovolumen </option>
                                  <option value={7}> Minibus </option>
                                  <option value={8}> Furgoneta </option>
                                  <option value={9}> Motocicleta </option>
                                </Input>
                  </FormGroup>
                  <Button type="submit">Crear nuevo vehiculo</Button>
                </Form>
```

Figura 63: Fragmento del componente VehicleCreator

Como vemos en la imagen, se utiliza un formulario para cumplimentar los campos: "*Matrícula del vehículo*", "*Nombre de la localización", "Latitud","Longitud"* y *"tipo de vehículo"*, utilizando este último para seleccionar entre los nueve distintos tipos de vehículos que tenemos.

Una vez cumplimentados los campos, se hace un *submit* y se llama a la función *createVehicle* que, al igual que la función para crear las sucursales, captura todos los valores del formulario y llama al proceso *createVehicle* de **vehicleTotal,** como se muestra en la siguiente imagen.

```
const createVehicleId = contract.methods["createVehicle"].cacheSend( vehicleNameLocation, vehicleLatitude, vehicleLatitudeExp, vehicleLongitude, vehicleLongitudeExp, vehicleName, vehicleType,btrue, this.props.branch_name,
  from: drizzleState.accounts[0], gas: 4712388,
  gasPrice: 100000000000
});

this.setState({ createVehicleId });
```

Figura 64: Fragmento del componente VehicleCreator

Al crear el vehículo, aparte de los valores capturados en el formulario, también se le añaden el nombre de la sucursal en el que ha sido creado y un valor de "*True*" para indicar que el vehículo está disponible.

Si se accede a este componente desde un lugar en el que no sea necesario la creación de vehículos, se llamará directamente al componente "**VehicleOrganize**" para continuar con el proceso de mostrar dichos vehículos.

7.3.8 VehicleOrganize

La finalidad de este componente es obtener todos los vehículos que están guardados en nuestra red Blockchain. Esto se consigue utilizando la función *map*, al igual que hemos hecho con el componente "***BranchCreator***".

```
var components = myVehicles.map((vehicle, index) => {
    if (((vehicle && vehicle.value) !== undefined)){


      return (

                    <ShowVehicles viewClient={this.props.viewClient}
                     account={this.props.account}
                      show_rented={this.props.show_rented}
                       adminselect={this.props.adminselect}
                        key={index}
                        dia={this.props.dia}
                        mes={this.props.mes}
                        anyo={this.props.anyo}
                        vehicleselect={this.props.vehicleselect}
                        client_view_pay={this.props.client_view_pay}
                        branch_to_pay={this.props.branch_to_pay}
                        branch_name={this.props.branch_name}
                         address={vehicle.value}
                          index={index}
                           drizzle={this.props.drizzle}
                            drizzleState={this.props.drizzleState}
                            />);
    }else return null;
});
```

Figura 65: Fragmento del componente VehicleOrganize

Como se puede comprobar en la imagen, se utiliza la función map para obtener los vehículos y se hace una pequeña comprobación en la cual, si no es "*undefined*"(ha sido creado de forma correcta), llamará al componente "**ShowVehicles**" para continuar con el proceso de muestra de vehículos. Si no es así, no llama a ningún componente ni muestra nada a través de la página Web.

7.3.9 ShowVehicles

El objetivo de este componente es mostrar u ocultar la información del vehículo, dependiendo de si se hace clic sobre el vehículo o no. Esto se consigue gracias a la función *showVehicle* que nos permite cambiar el estado de la variable "showVehicle" entre "*True*" y "*False*" para mostrar u ocultar la información del vehículo.

```
showVehicle(e){
    if(e) e.preventDefault();
    let showVehicle;
    if (this.state.showVehicle) {
        showVehicle=false;
        this.setState( {showVehicle});
    }else{
        showVehicle=true;
        this.setState( {showVehicle});
    }
}
```

Figura 66: Fragmento del componente ShowVehicles

Como se muestra en la imagen, la función es un *toggle* del valor del estado "showVehicle" en el cual, si tiene un valor de "*True*" pasa a valer "*False*" y viceversa.

Por último, hay que mencionar que este componente llama al componente "**VehiclesShow_2**", que es el encargado de mostrar los vehículos y que interactúa con la función que acabamos de explicar. Esto último se entenderá mejor cuando se explique el componente "**VehiclesShow_2**".

7.3.10 VehiclesShow_2.

La utilidad de este componente es muy amplia. Sirve para mostrar la información del vehículo de forma visual en modo lista, o en modo QR, indicando únicamente la matrícula del vehículo y la localización de este en el segundo caso. Dependiendo desde donde se acceda a este componente, también permite el alquiler y la devolución el vehículo, abonando un coste por el tiempo que ha sido alquilado.

Vamos a empezar explicando que la funcionalidad básica que se aplica, independientemente desde donde se acceda, es obtener todos los parámetros que se encuentran dentro de los contratos tipo **Vehicle** (que se han creado previamente), y mostrarlos por pantalla.

Hemos mencionado que se muestran por pantalla unos datos u otros dependiendo desde donde se acceda, pero vamos a especificar a qué nos referimos:

- Si se accede a este componente desde la pestaña de "Administrador" no se mostrará ningún vehículo, debido a que está diseñado para que únicamente se puedan ver las sucursales, a fin de no mostrar demasiada información en una sola pantalla.
- Si se accede desde la ventana de "Sucursales" mostrará todos los vehículos que tengan creados las sucursales, pero no se podrá ni alquilar ni devolver desde esa pestaña. Esto es así porque esa pestaña únicamente la verán los administradores.
- Si se accede desde "Vehículos" se mostrarán todos los vehículos que se puedan alquilar y, desde esa misma pestaña, se podrá realizar el alquiler.
- Si se hace desde la pestaña "Clientes" se mostrarán todos los vehículos que el cliente tenga alquilados, y también se podrán devolver.

Ha sido necesario realizar una breve explicación de todo ello porque es este componente el que gestiona todas esas opciones, pudiendo mostrar los vehículos o no hacerlo, y también poder alquilarlos o devolverlos.

Vamos a explicar de forma resumida las funciones que interactúan en este componente, que son varias.

- La función *Show_rented* se utiliza para mostrar el botón de "Alquilar vehículo". Una vez que se haya pulsado llamará a la función *rented_vehicle* que empezará con el proceso de alquiler de vehículo.

```
Show_rented(){
  if(this.state.showrented === false){
    return (<Button outline color="success" onClick={this.rented_vehicle} >Alquilar vehiculo</Button>)
  }else return(null)
  ;
}
```

Figura 67: Fragmento del componente VehiclesShow_2

- La función *Show_unrented* mostrará el botón de "Pagar y devolver", mostrando la cantidad que hay que abonar para poder devolverlo. Una vez pulsado, llamará a la función *handleTansfer* que comenzará el proceso de abono del alquiler y devolución del vehículo.

```
Show_unrented(amount){

  if(this.state.showrented === false){
    return (
    <div>
      <p>Usted tiene que pagar {amount}</p>
    <Button outline color="danger" onClick={this.handleTransfer.bind(this, amount)} >Pagar y devolver vehiculo</Button>
    </div>)
  }else return(null)
  ;
}
```

Figura 68: Fragmento del componente VehiclesShow_2

- La función *rented_vehicle* se utiliza para modificar el contrato del vehículo, pasándole los parámetros de la dirección del contrato del vehículo, la dirección de la cuenta del cliente y la fecha en la que empieza a alquilar el vehículo.

```
rented_vehicle()
{
 const {drizzle, drizzleState} = this.props;
 const VehicleTotal = drizzle.contracts.VehicleTotal;
  const VehicleContractId = VehicleTotal.methods["rented_vehicle"].cacheSend(this.props.address,this.props.account,this.props.dia, this.props.mes,this.props.anyo ,{
    from: drizzleState.accounts[0], gas: 4712388,
    gasPrice: 100000000000
  });
let flag = false;

this.setState({ VehicleContractId,flag });
```

Figura 69: Fragmento del componente VehiclesShow_2

- La función *handleTransfer* se utiliza para hacer la transacción de tokens, en este caso *ether*, entre el cliente y la sucursal. Esto es posible gracias a que se captura del contrato la dirección de la Wallet de la sucursal. También se captura la dirección del cliente en el componente "**GlobalContainer",** y estos dos datos nos posibilitan hacer la transferencia de tokens.

```
handleTransfer(amount) {
  const from = this.props.account;
  const to = this.props.branch_to_pay;
  const value = this.web3.toWei(amount, "ether");
  const txnHash = this.web3.eth.sendTransaction({ from, to, value });
  const handle = setInterval(() => {
      const txn = this.web3.eth.getTransaction(txnHash);
      if (txn) {
          clearInterval(handle);
          this.refreshUI();
          this.unrented_vehicle();
      }
  }, 100);
```

Figura 70: Fragmento del componente VehiclesShow_2

La transferencia es de una cantidad que la determina la variable "amount", y se calcula de la siguiente manera:

```
let amount = ((Number(this.props.anyo) - Number(anyo))*1 + (Number(this.props.mes) - Number(mes))*0.4 + (Number(this.props.dia) - Number(dia))*0.014);
```

Figura 71: Fragmento del componente VehiclesShow_2

Se hace una resta de la fecha actual con la fecha del día de alquiler, haciendo que cada año cueste 1 ether, cada mes cueste 0.4 ether, y cada día cueste 0.014 ether.

Por último, esta función llama a otras dos funciones, que son *refeshUI* y *unrented_vehicle*.

- La función *refeshUI* se utiliza únicamente para comprobar el balance de la cuenta una vez realizada la transacción.

```
refreshUI() {

  const accounts = this.web3.eth.accounts.map(account => {
      let balance = this.web3.eth.getBalance(account);
      balance = this.web3.fromWei(balance, "ether").toNumber();
      return { account, balance };
  });
  this.setState({ accounts });
}
```

Figura 72: Fragmento del componente VehiclesShow_2

- La función *unrented_vehicle* se utiliza para modificar el contrato del vehículo, pasándole los parámetros de la dirección del contrato, las coordenadas y la dirección de la localización del vehículo.

```
unrented_vehicle()
{
  const {drizzle, drizzleState} = this.props;
  const VehicleTotal = drizzle.contracts.VehicleTotal;
  const instance = drizzle.contracts[this.props.address];


  const VehicleContractId = VehicleTotal.methods["unrented_vehicle"].cacheSend(this.props.address,this.state.vehicleLatitude,this.state.vehicleLatitudeExp,this.state.vehicleLongitude,this
    from: drizzleState.accounts[0]

});
let flag = false;
this.setState({ VehicleContractId,flag });
```

Figura 74: Fragmento del componente VehiclesShow_2

Al realizar estas operaciones se actualizará el estado del vehículo a “disponible”, y se mostrarán las nuevas coordenadas introducidas que corresponderán a la ubicación donde se encuentra el vehículo en ese momento.

- La función *changelocalitation* recoge del formulario la nueva localización del vehículo, y lo guarda en las variables de estado correspondientes para que, posteriormente puedan ser utilizadas cuando se invoque a la función *unrented_vehicle*.

- Las funciones *showVehicle* y *button_devolver* son dos funciones, cuyo propósito es hacer un *toggle* del valor de un estado. Hemos visto ya varios ejemplos de estos a la hora de explicar los componentes, así que no volveremos a profundizar en ellos.

```
button_devolver(e){
  if(e) e.preventDefault();
  let button_devolver;
  if (this.state.button_devolver) {
    button_devolver=false;
      this.setState( {button_devolver});
  }else{
    button_devolver=true;
      this.setState( {button_devolver});
  }
```

```
showVehicle(e){
  if(e) e.preventDefault();
  let showVehicle;
  if (this.state.showVehicle) {
      showVehicle=false;

      this.setState( {showVehicle});
  }else{
      showVehicle=true;

      this.setState( {showVehicle});
  }
}
```

Figura 75 y 76: Fragmento del componente VehiclesShow_2

A continuación, vamos a mostrar un caso práctico de cómo se muestra la información de los vehículos en la vista del cliente. Cada caso es distinto, pero todos siguen la misma estructura.

```
<Media>
  <Media left href="#">
    <Media object src={imageurl} alt="" onClick={this.showVehicle} className="pointer rounded-circle"/>
  </Media>
  <Media body className="marginLeft">
    <Media heading onClick={this.showVehicle} className="">
    <h2 className="subrayar">{"Vehiculo "+(vehicle_registration)+". "}</h2>
    </Media>
    <Row>
        <Col>
    <ul>
    <li>Se sitúa en: {locationName}.  <a target="_blank" rel="noopener noreferrer" href={"https://www.google.com/maps/search/?api=1&query="+String(latitude)+","+String(longitude)}>Mostr
      <li>Matricula vehiculo: {vehicle_registration}</li>
      <li>Es un vehiculo tipo: {vehicleType}</li>
      </ul>
      </Col>
        <Col>
        <iframe name="gMap" width="100%" height="200" frameborder="0" scrolling="no" marginheight="0" marginwidth="0" src={`https://www.google.com/maps/embed/v1/place?q=${String(latitud
        </Col>
        </Row>

      <Row>
      {this.Show_unrented(amount)}
      <Button outline color="info" onClick={() => window.location.reload(false)}>Confirmar cambios</Button>
      </Row>
  </Media>
</Media>
```

Figura 77: Fragmento del componente VehiclesShow_2

Como se puede apreciar en la imagen, se muestra en una lista la información del vehículo y también es mostrada, tanto un *link* como un *iframe,* con la posibilidad de ver dónde se encontraba el vehículo que habían recogido.

Esta se podrá ver de forma comprimida por medio de un código QR, pero por este medio únicamente se podrá visualizar la ubicación del vehículo en caso de escanear dicho código. Para ello, se utilizará este fragmento de código.

```
<Container>
<Container className="pointer margins" onClick={this.showVehicle} fluid>
<Row>
        <Col md="4" sm="12">
            <QRCode value={"https://www.google.com/maps/search/?api=1&query="+String(latitude)+","+String(longitude)}/>
        </Col>
        <Col md="8">
        <h2 className="subrayar">{"Vehiculo "+(vehicle_registration)+". "}</h2>
        </Col>
  </Row>
</Container>
<Container>
```

Figura 78: Fragmento del componente VehiclesShow_2

En el supuesto de querer devolver el vehículo, aparecerá un formulario que se tendrá que rellenar para confirmar su devolución.

En el caso de acceder desde la pestaña de la sucursal, no se podrán ver las opciones de alquilar o devolver el vehículo como se ha mencionado anteriormente. Por último, en el caso de acceder desde la pestaña de vehículos, únicamente se podrá alquilar, nunca devolver desde esa misma pestaña. Eso es así, debido a que los vehículos alquilados no aparecen en esa pestaña.

7.3.11 Registration

Este componente se utiliza para mostrar la información de las opciones de alquiler de los vehículos que tienen los clientes una vez registrados, o si se accede desde la pestaña de administración, permitirá mostrar los clientes del sistema.

En el primer caso, se mostrará información genérica de los vehículos, llamando al componente "**Carousel_view"**. Aunque en ambos casos se llamará al componente "**RegistrationManager"**, que es el encargado de comenzar con los pasos necesarios para obtener y/o mostrar los clientes del sistema.

7.3.12 Carousel_view

La finalidad de este componente es mostrar la información en carrusel y hacerlo de forma cíclica, como son las imágenes de los distintos tipos de vehículos con un pequeño texto descriptivo de estos. Esto se consigue instalando el paquete *react-bootstrap-carousel* que permite incorporar estos efectos.

A continuación, se expone una pequeña parte del código en la que se muestra cómo se ha implementado. Hay que destacar, que previamente ha habido que generar una serie de funciones para poder desplazar las imágenes de derecha a izquierda y viceversa.

```
<div style={{ height: 400,backgroundColor: "Wheat" }}>
<div className="carousel-center">
  <img
    style={{ minWidth: '100%',minHeight: "100%" }}
    src="img/car/coche1.png"
  />
  </div>
  <div style={{color: "black"}} className="carousel-caption"><p><u>Vehiculo Economico</u></p>
  <p>Cómodo y practico, ideal para moverse por la ciudad</p></div>
</div>
```

Figura 79: Fragmento del componente Carousel_view

7.3.13 RegistrationManager

El objetivo de este componente es el mismo que el de los componentes "**BranchCreator"** y "**VehicleOrganize"**. Se utiliza este componente para que, por medio de la función *map*, podamos obtener los clientes de la red Blockchain donde están almacenados.

```
var components = myregistration.map((clientes, index) => {
    if (((clientes && clientes.value) !== undefined)){

        if(this.props.vehicleclick)//modo visializar vehiculos
        {
      return (
                <div>
                    <Registration_show2
                     account={this.props.account}
                      show_rented={this.props.show_rented}
                       adminselect={this.props.adminselect}
                       vehicleclick={this.props.vehicleclick}
                        key={"client"+index}
                        dia={this.props.dia}
                        mes={this.props.mes}
                        anyo={this.props.anyo}
                        branch_to_pay={this.props.branch_to_pay}
                        branch_name={this.props.branch_name}
                         address={clientes.value}
                          index={index}
                           drizzle={this.props.drizzle}
                            drizzleState={this.props.drizzleState}
                            /></div>);
    }
```

Figura 80: Fragmento del componente RegistrationManager

Este componente llama a tantos componentes "**Registartion_show2"** como clientes haya registrados en el sistema. Hay dos formas de acceder a este componente, aunque la finalidad de ambas es la misma. La única diferencia es que accediendo por medio de la pestaña de administración se muestran los vehículos, y accediendo desde la pestaña de vehículos se comprueba que el cliente previamente se ha registrado, para permitirle acceder al menú de alquiler de los vehículos.

7.3.14 Registration_show2

La utilidad de este componente es conseguir la información que se almacena dentro de los contratos de los clientes. Al igual que la mayoría de los componentes descritos hasta ahora, dependiendo desde donde se realice el acceso, se mostrarán los datos del cliente o por ende no mostrará nada, y únicamente se utilizará la información del cliente para comprobar que está registrado en el sistema.

En el caso de acceder desde la pestaña de administrador, se mostrará una lista de los clientes. Por defecto, se mostrará con un código QR, que al escanearlo permitirá visualizar la dirección de la Wallet que está asociada a ese cliente. En el caso de querer ver la información de forma más extensa, haciendo clic, se mostrará una lista con la información del "nombre", "apellidos", "DNI u otro documento identificativo" y "Wallet Asociada". Todo esto se puede apreciar en la siguiente imagen.

```
if(this.state.showRegistration){

    return (
      <Media>
        <Media left href="#">
          <Media object src="img/icons/admin.png" alt="" onClick={this.showRegistration} className="pointer rounded-circle"/>
        </Media>
        <Media body className="marginLeft">
          <Media heading onClick={this.showRegistration} className="pointer subrayar">
          {"Cliente número "+(this.props.index + 1)+": "+registrationName+ " "+registrationSurname}
          </Media>
          <ul>
            <li>DNI: {registrationDNI}</li>
            <li>Wallet asociada: {registrationAddresss}</li>
          </ul>

        </Media>
      </Media>
    );
}else{
    return (

        <Container className="pointer margins" onClick={this.showRegistration} fluid>
        <Row>
              <Col md="3" sm="12">
                  <QRCode value={this.props.address}/>
              </Col>
              <Col md="9">
                  <h2 className="subrayar">{"Cliente número "+(this.props.index + 1)+": "+registrationName+ " "+registrationSurname}</h2>
              </Col>
          </Row>
        </Container>);
}
```

Figura 81: Fragmento del componente Registration_show2

Como en todos los casos anteriores, se utiliza una función para hacer un *toggle* del estado de una variable que permita mostrar un contenido u otro. En este caso la función se llama "**showRegistration",** que al ser similar que resto de funciones que desempeñan la misma finalidad, no pondremos su código para evitar redundar información.

En caso de acceder desde la pestaña de vehículos, únicamente se utilizarán los datos suministrados para saber si el cliente que se encuentra actualmente en la página Web está

registrado o no y, en caso afirmativo, se mostrará la lista de posibles vehículos que están disponibles. Esto último se realizará en el componente "**Vehicle"**, que será invocado en el caso de que se acceda a este componente desde la pestaña de Vehículos.

7.3.15 Vehicle

Utilizamos este componente para que, si el cliente está registrado en el sistema, muestre una lista completa de los países de la Unión Europea donde se encuentren sucursales que tengan vehículos disponibles para alquilar. Si el cliente aún no está registrado en el sistema, no mostraría nada, evitando así que una persona sin estar registrada pudiese alquilar algún vehículo.

Cuando se decida activar el icono de un país para comprobar la disponibilidad de vehículos, aparecerá una lista con las sucursales de ese país junto a la localización de donde se encuentran. Desde ahí, el cliente podrá navegar hasta encontrar el vehículo que desee. Todo esto se consigue gracias a un gran número de funciones *toggle* que permiten variar lo mostrado por pantalla, dependiendo donde se pulse.

Hay un total de veintisiete funciones que implementan un *toogle* del estado de una variable, una por cada país, como se muestra en la siguiente imagen.

```
this.showCountrie1 = this.showCountrie1.bind(this);
this.showCountrie2 = this.showCountrie2.bind(this);
this.showCountrie3 = this.showCountrie3.bind(this);
this.showCountrie4 = this.showCountrie4.bind(this);
this.showCountrie5 = this.showCountrie5.bind(this);
this.showCountrie6 = this.showCountrie6.bind(this);
this.showCountrie7 = this.showCountrie7.bind(this);
this.showCountrie8 = this.showCountrie8.bind(this);
this.showCountrie9 = this.showCountrie9.bind(this);
this.showCountrie10 = this.showCountrie10.bind(this);

this.showCountrie11 = this.showCountrie11.bind(this);
this.showCountrie12 = this.showCountrie12.bind(this);
this.showCountrie13 = this.showCountrie13.bind(this);
this.showCountrie14 = this.showCountrie14.bind(this);
this.showCountrie15 = this.showCountrie15.bind(this);
this.showCountrie16 = this.showCountrie16.bind(this);
this.showCountrie17 = this.showCountrie17.bind(this);
this.showCountrie18 = this.showCountrie18.bind(this);
this.showCountrie19 = this.showCountrie19.bind(this);
this.showCountrie20 = this.showCountrie20.bind(this);

this.showCountrie21 = this.showCountrie21.bind(this);
this.showCountrie22 = this.showCountrie22.bind(this);
this.showCountrie23 = this.showCountrie23.bind(this);
this.showCountrie24 = this.showCountrie24.bind(this);
this.showCountrie25 = this.showCountrie25.bind(this);
this.showCountrie26 = this.showCountrie26.bind(this);
this.showCountrie27 = this.showCountrie27.bind(this);
```

Figura 82: Fragmento del componente Vehicle

A cada función, se le asigna una variable, que por defecto está con un valor "*False*". En el momento en el que alguna de las variables cambie su valor a "*True*", se dejarán de mostrar todos los países y se mostrará únicamente la lista de las sucursales del país seleccionado. Esto último se consigue asignando a cada país un valor del 1 a 27. Se logrará de esta forma que, una vez seleccionado un país, únicamente se muestren las sucursales que hagan coincidir el valor de su campo con el valor del país.

```
<div>
<Media>
  <Media left href="#">
    <Media object src="img/countries/chipre.png" alt="" className="pointer rounded-circle" onClick={this.showCountrie6}/>
    </Media>
    <Media body className="marginLeft">
          <Media heading className="pointer subrayar">
           {"Chipre"}
           </Media>
   </Media>
 </Media>
 <Adminn dia={this.props.dia}
                  mes={this.props.mes}
                  anyo={this.props.anyo}
                  drizzle={this.props.drizzle}
                  drizzleState={this.props.drizzleState}
                  account={this.props.account}
                  branch_create_view={false}
                  client_view_pay={true}
                  adminselect={false}
                  vehicleselect={true}
                  countrie_number={5}
          />
</div>
```

Figura 83: Fragmento del componente Vehicle

En esta imagen se puede apreciar el contenido, que se mostrará por pantalla, si por ejemplo se seleccionase el país de **Chipre** como objetivo del alquiler del vehículo.

Donde las funciones *toggle* tendrán los mismos cometidos que han sido referenciados anteriormente, por tanto omitimos su explicación de nuevo.

7.3.16 NEWClient

La finalidad de este componente es la de registrar a los clientes para que puedan acceder a los vehículos. Ha sido necesario crear este componente, para hacer posible enlazar la Wallet que alquila el vehículo con una persona.

Para que los clientes puedan registrase, deberán cumplimentar un formulario con los campos: "Nombre", "Apellidos" y "DNI".

```
  <Container>

  <h3 className="margins">Hay un total de {(clientLength && clientLength.value) || 0} clientes registrados.</h3>
<Row>
  <Col md={4}>
  <Form onSubmit={this.createClient.bind(this)} className="formBox">
    <FormGroup>
      <Label>Nombre </Label>
      <Input type="text" id="clientName" placeholder="Nombre" required />
    </FormGroup>
    <FormGroup>
      <Label>Apellidos </Label>
      <Input type="text" id="clientSurname" placeholder="primero, segundo" required />
    </FormGroup>
    <FormGroup>
      <Label>DNI o equivalente</Label>
      <Input type="text" id="clientDNI" placeholder="06666666E" required />
    </FormGroup>
    <Button type="submit">Registrarse</Button>
  </Form>
  </Col>
  <Col md={8}>
<div>
<h2>Información:</h2>
<h5>Es necesario rellenar todos los campos para poder registrarse de forma correcta.</h5>
<ul>
<li><u>Nombre:</u>Introduzca el nombre que aparece en su DNI u otro documento identificativo.</li>
<li><u>Apellidos:</u> Introduzca los apellidos que aparecen en su DNI u otro documento identificativo.</li>
<li><u>DNI:</u> Introduzca su DNI u otro documento identificativo.</li>
</ul>
```

Figura 84: Fragmento del componente NEWClient

Este es el código que genera el formulario que acabamos de comentar. En él se pueden apreciar cómo los campos mencionados son de cumplimentación obligatoria.

Una vez rellenados los campos, se llamará a la función *createClient* que tomará los datos introducidos en el formulario. Una vez capturados, se llamará al proceso *createCLient* del contrato **ClientTotal** y se les pasarán los parámetros del formulario junto a la dirección de la Wallet desde la que se está creando el cliente, para así poder asociarla.

```
const createClientId = contract.methods["createCLient"].cacheSend( clientName, clientSurname, clientDNI,this.props.account, {
  from: drizzleState.accounts[0], gas: 4712388,
  gasPrice: 100000000000
});
```

Figura 85: Fragmento del componente NEWClient

7.3.17 Client

Este es el componente que se ve por defecto al acceder por primera vez a nuestro portal web. En este componente se muestra toda la información básica que deben saber nuestros futuros clientes o nuestros clientes actuales.

```
          <Container fluid>
    <Row>
    <Col style={{ minHeight: 200, backgroundColor: "Wheat" }}>
          <p class=" h-100 d-flex justify-content-center align-items-center" >Usted está utilizando la cuenta: {this.props.account}</p>
    </Col>
          <Col style={{backgroundColor: "Wheat" }} class= "text-center">
          <p class=" h-100 d-flex justify-content-center align-items-center" >Si quiere poder probar nuestro servicio, necesitará  registrasrse previamente en le sistema. Esto es debido a r
            Registrese y pruebe nuestros servicios que se encuentran en toda Europa.
Una vez registardo, vaya a la pestaña de vehicuo para seleccionar el vehiculo que desee alquilar
          </p>
          </Col>
          </Row>
          </Container>
           <h5 class=" h-100 d-flex justify-content-center align-items-center"><u>Lista de vehiculos que tiene alquilado en este momento:</u></h5>



          <Adminn dia={this.props.dia} mes={this.props.mes} anyo={this.props.anyo} drizzle={this.props.drizzle} drizzleState={this.props.drizzleState} account={this.props.account} branch_cr
          />


          </div>
          </Container>
```

Figura 86: Fragmento del componente Client

En el caso de que el cliente aún no esté registrado, únicamente verá la información básica que ofrece nuestra Web. En cambio un cliente ya registrado, también podrá observar la lista de vehículos que actualmente se encuentren alquilados. Esto último se conseguirá llamando al componente "**Adminn",** que será el encargado de comenzar con todo el proceso de obtención de los vehículos y la visualización de estos.

Esta imagen es un fragmento de código en la que se puede apreciar cómo se llama al componente "**Adminn"** y cómo muestra una parte de la información para futuros clientes.

7.3.18 Selection

Este componente es esencial para un correcto funcionamiento de nuestra Web. Se trata del componente que nos permite cambiar entre pestañas. Esto se consigue gracias a una serie de funciones descritas en el componente "**GlobalContainer***",* que modifica el estado de una variable entre "*admin*", "*vehicle*", "*client*", "*NEWclient*" y "*Registration*".

Este componente es el que utiliza las funciones implementadas que han sido mencionadas anteriormente, cambiando su valor cada vez que se pulse el texto que indica la nueva pestaña que se quiera visitar. Esto se consigue con el siguiente código.

```
<Container fluid>
    <Navbar color="lightsalmon" light expand="md">
    <div id="pequena">
      <NavbarBrand href="#"><img src="img/icons/Cliente.png" alt="Manipulator View" id="pequena"/></NavbarBrand>
      </div>
      <Nav className="ml-auto" navbar>

      <Container fluid className="text-center justify-content-md-center"><h1 c  id="centerio">Zona para Registrarse</h1></Container>

      <NavItem>
      <Nav className="ml-auto" navbar>
        <NavItem>
          <NavLink href="#" className="text-info" onClick={this.props.changeViewClient}><h7  class="un">Cliente</h7></NavLink>
        </NavItem>
        <NavItem>
          <NavLink href="#" className="text-info" onClick={this.props.changeViewAdmin}><h7  class="un">Administrador</h7></NavLink>
        </NavItem>
        <NavItem>
          <NavLink href="#" className="text-info" onClick={this.props.changeViewManip}><h7  class="un">Vehiculos</h7></NavLink>
        </NavItem>
        <NavItem>
          <NavLink href="#" className="text-info" onClick={this.props.changeViewAdminBranches}><h7  class="un">Sucursales</h7></NavLink>
        </NavItem>
        </Nav>

        </NavItem>
        </Nav>
    </Navbar>
    </Container>
);
```

Figura 87: Fragmento del componente Selection

En la imagen se puede apreciar cómo cada texto indicativo de una pestaña tiene asociada una función, cuyo objetivo es cambiar el estado de la variable del componente "**GlobalContainer***"* al estado adecuado para visualizar la pestaña deseada. Este código es necesario replicarlo por cada distinta pestaña a la que se quiera acceder desde este menú.

7.3.19 Botton

Este último componente se utiliza para mostrar en nuestra página Web un fragmento de texto característico en todas las páginas que suelen traer información genérica. Este componente tiene una utilidad meramente estética, debido a que únicamente se utiliza para hacer más

vistosa a la página Web. Por ello se va a poner a continuación, parte del código del componente que realiza la acción visual descrita con anterioridad.

```
<Col className="newsletter" md="2">
    <div className="header">
        <h6 class="un">Contactar</h6>
        <ul className="contact list-unstyled">
        <li><a href="/" onClick={e => {if(e) e.preventDefault();}}>Ayuda/Preguntas frecuentes </a> </li>
        <li><a href="/" onClick={e => {if(e) e.preventDefault();}}>Socios</a></li>
        <li><a href="/" onClick={e => {if(e) e.preventDefault();}}>Anunciate con nosotros</a></li>
    </ul>

    </div>
</Col>
<Col className="newsletter" md="2">
    <div className="header">
        <center><h6 class="un">Más</h6></center>
        <div className="text-center" >Valoración de 77 sobre 100</div>
        <br></br>
        <Progress color="warning" value={77} />
```

Figura 88: Fragmento del componente Botton

7.4 Segmentación de página Web por componentes.

Una vez explicados todos los componentes por separado, vamos a continuar mostrando por medio de capturas de nuestra Web, cómo se comportan los componentes en conjunto. Se indicará cuáles son los que se utilizan en cada página por medio de cuadros de colores, donde se incluyen los componentes específicos que interactúan en ese espacio. El único que no se encuentra dentro de su rectángulo es "**GlobalContainer**", debido a que engloba a todos.

Los componentes que interactúan son:

- **GlobalContainer:** es el componente esencial que actúa como contenedor del resto de componentes.
- **Selection:** componente utilizado para desplazarse entre las distintas pestañas.
- **Client:** muestra la información del marco superior de la página Web.
- **Adminn**: componente utilizado para administrar las sucursales que contienen los vehículos.
- **BranchCreator:** utilizado para obtener de la red Blockchain los contratos de las sucursales.
- **BranchsShow:** utilizado para acceder a las sucursales que contienen los vehículos.
- **VehicleCreator:** componente utilizado para crear los distintos tipos de vehículos.
- **VehicleOrganize**: utilizado para acceder a los vehículos.
- **ShowVehicles:** utilizado para acceder a los vehículos.
- **VehicleShow_2:** utilizado para acceder a los vehículos.
- **Botton:** componente utilizado para mostrar la información en el marco inferior de página Web.
- **NewClient:** componente utilizado para registrar a los nuevos clientes en el sistema.
- **Registration:** componente esencial para mostrar la información de los tipos de alquileres disponibles.
- **Carousel_view:** componente necesario para mostrar los tipos de vehículos de que dispone nuestra empresa.

- **RegistartionManager:** utilizado para obtener, de la red Blockchain, los contratos de los clientes registrados.
- **Registration_show2:** componente empleado en mostrar la información de los clientes del sistema.
- **ShowCountries:** componente necesario para mostrar la información de las sucursales dependiendo del país que sean.

7.4.1 Página *Cliente*:

En la siguiente imagen encontramos la página de clientes, donde se puede apreciar cómo interactúan los distintos componentes para mostrar la página en cuestión.

En esta página se pueden ver los vehículos que el cliente tiene alquilados en ese momento y, desde ella, también es posible devolver y abonar los vehículos alquilados.

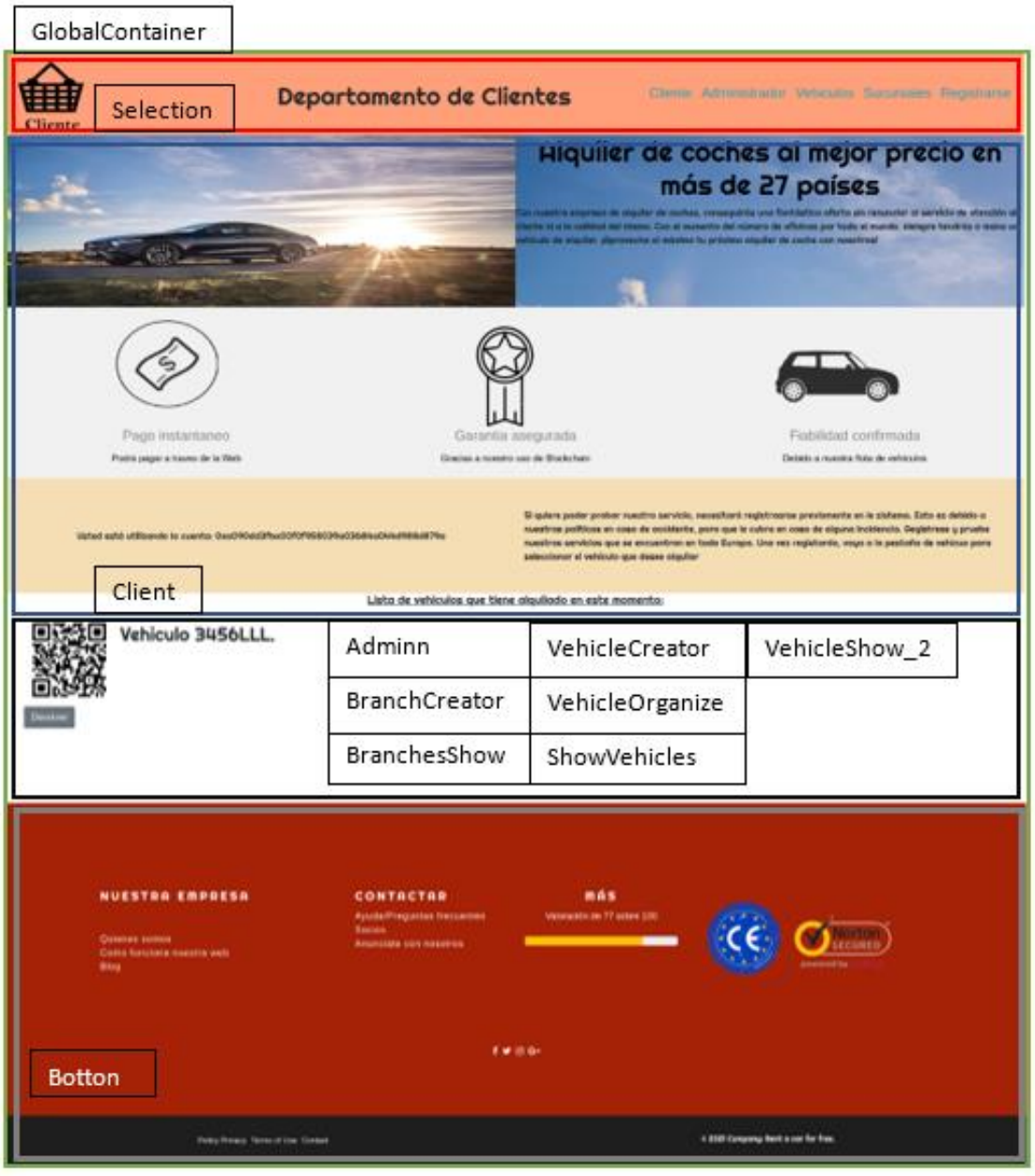

Figura 89: Página Cliente con los componentes que se compone

7.4.2 Página *Administrador*:

En la siguiente imagen encontramos la página de administrador, donde se puede apreciar cómo interactúan los distintos componentes para mostrar dicha página.

En ella se pueden crear las sucursales, para lo cual habrá que cumplimentar el formulario, y también es posible visualizar todas las sucursales del sistema.

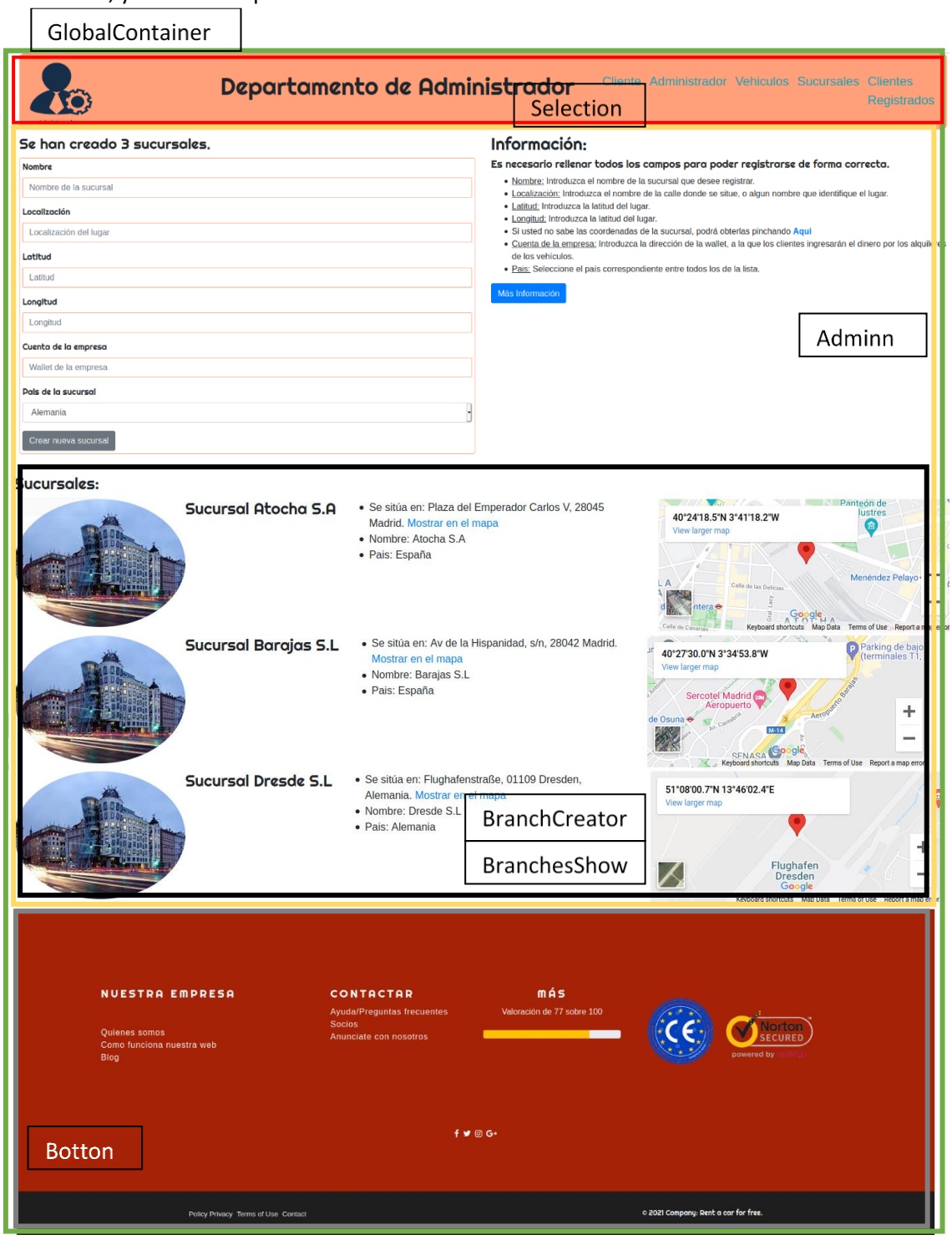

Figura 90: Página Administrador con los componentes que se compone

7.4.3 Página *Vehículos*:

En la siguiente imagen encontramos la página de vehículos, donde se puede apreciar cómo interactúan los distintos componentes para poder mostrarla.

En esta página se permite, una vez registrado el cliente, observar todos los países donde se encuentra la empresa operativa y se podrá alquilar el vehículo desde la sucursal elegida.

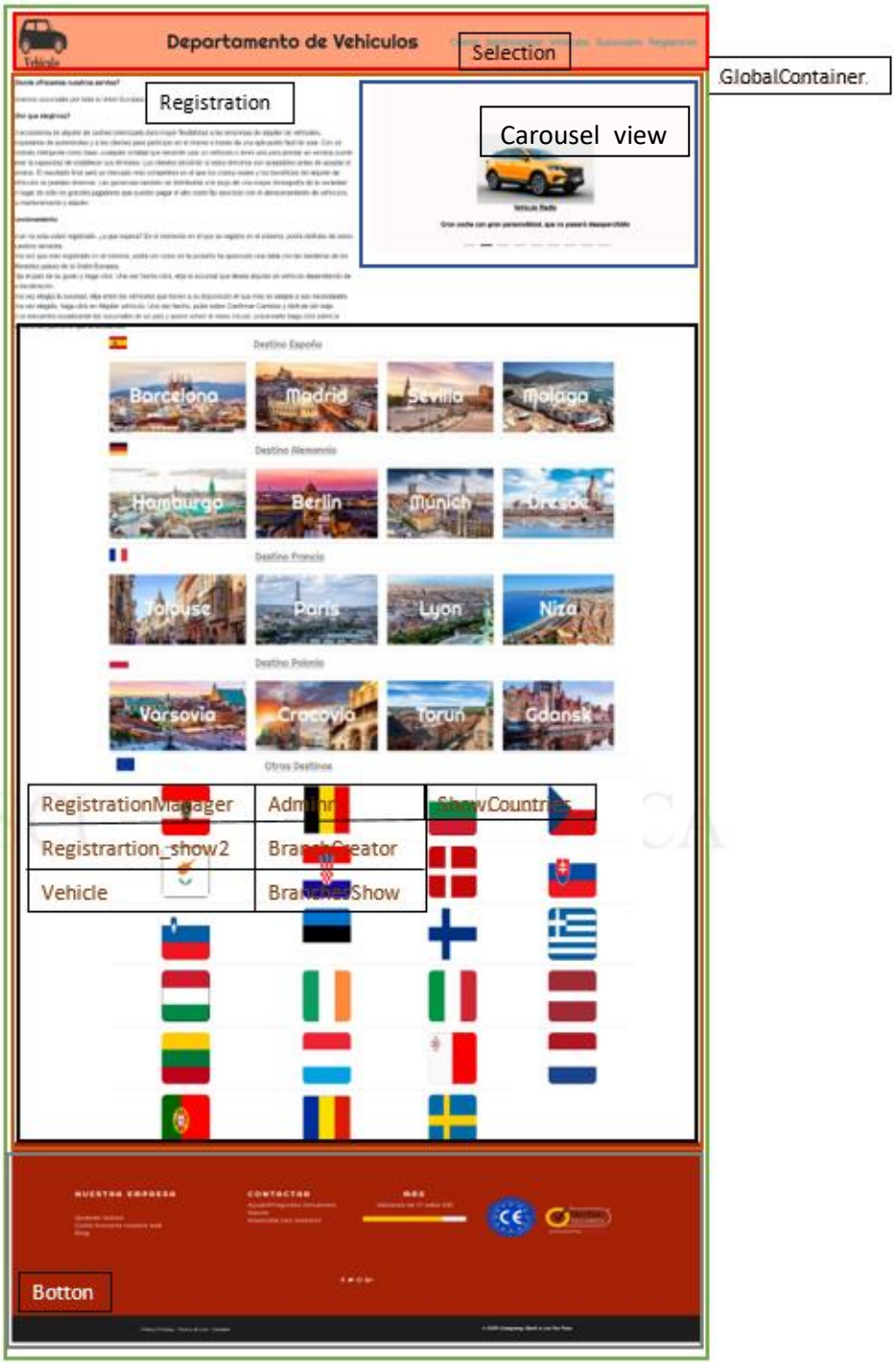

Figura 91: Página Vehículos con los componentes que se compone

La zona recuadrada como **“GlobalContainer”**, **“Selection”**, **“Botton”** y **“Registration”**, se mantienen sin cambios.

Si se hace clic sobre un destino o en un país en concreto, se mostrará la imagen expuesta a continuación. Hay que resaltar que la bandera y el nombre que aparecen, son distintos para cada país.

Figura 92: Fragmento de página Vehículos con los componentes que se compone

7.4.4 Página *Sucursales*:

En la siguiente imagen encontramos la página de sucursales, donde se puede apreciar cómo interactúan los distintos componentes para mostrar la página en cuestión.

En esta página se aprecian todas las sucursales creadas del sistema y, cumplimentando el formulario correspondiente, se pueden crear los vehículos en las sucursales.

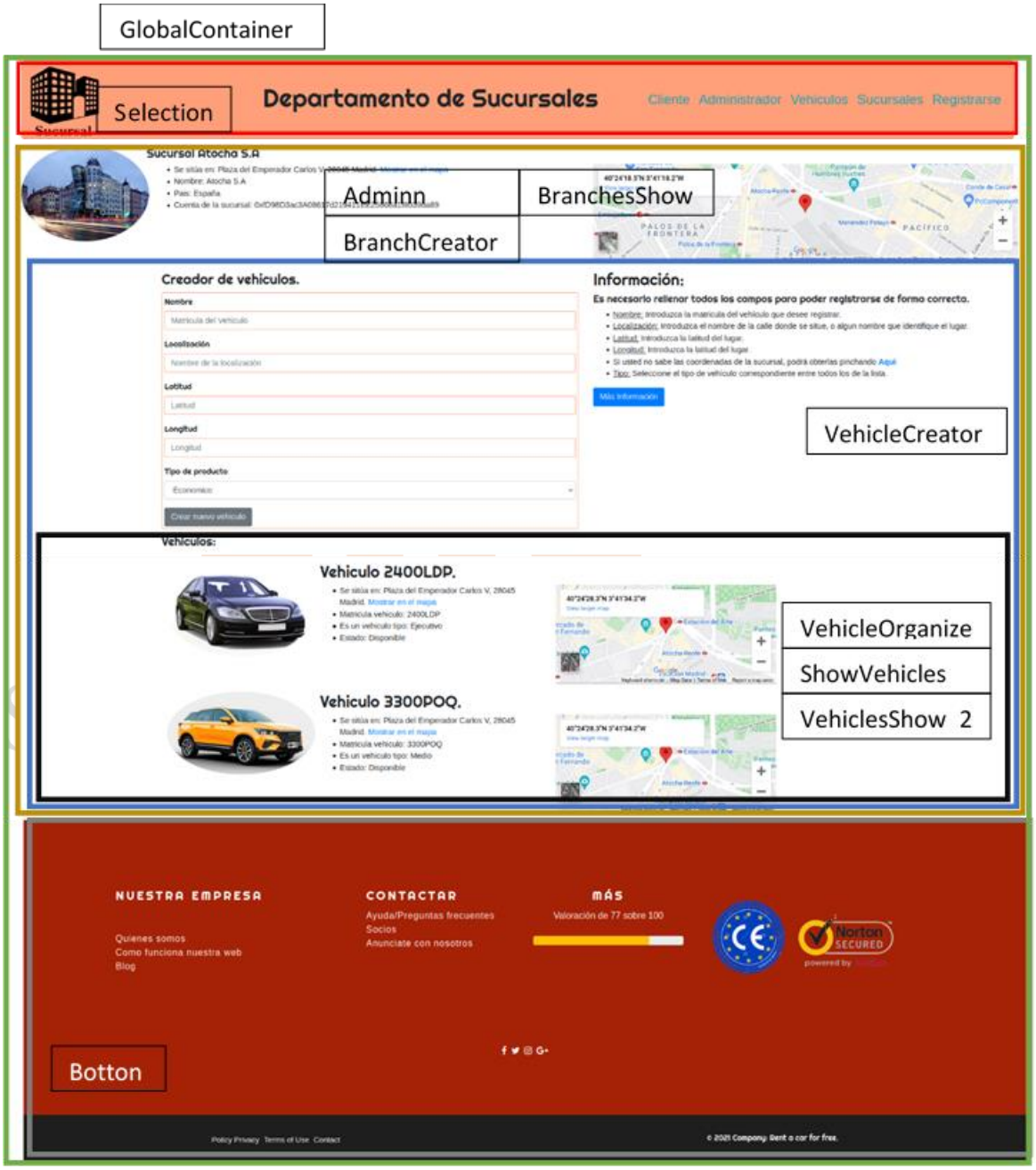

Figura 93: Página Sucursales con los componentes que se compone

7.4.5 Página *Registrarse*:

En la siguiente imagen encontramos la página para registrarse en el sistema, donde se puede apreciar cómo interactúan los distintos componentes para mostrarla.

En esta página los clientes, al rellenar el formulario de forma correcta, pueden registrarse en el sistema para poder acceder a todas las funciones de la página Web.

Figura 94: Página Registrarse con los componentes que se compone

7.4.6 Página *Clientes Registrados*:

En la siguiente imagen encontramos la página para poder ver los clientes registrados en el sistema, donde se puede apreciar cómo interactúan los distintos componentes para mostrar la página en cuestión.

En esta página el administrador puede visualizar toda la información de los clientes que se encuentran registrados en el sistema.

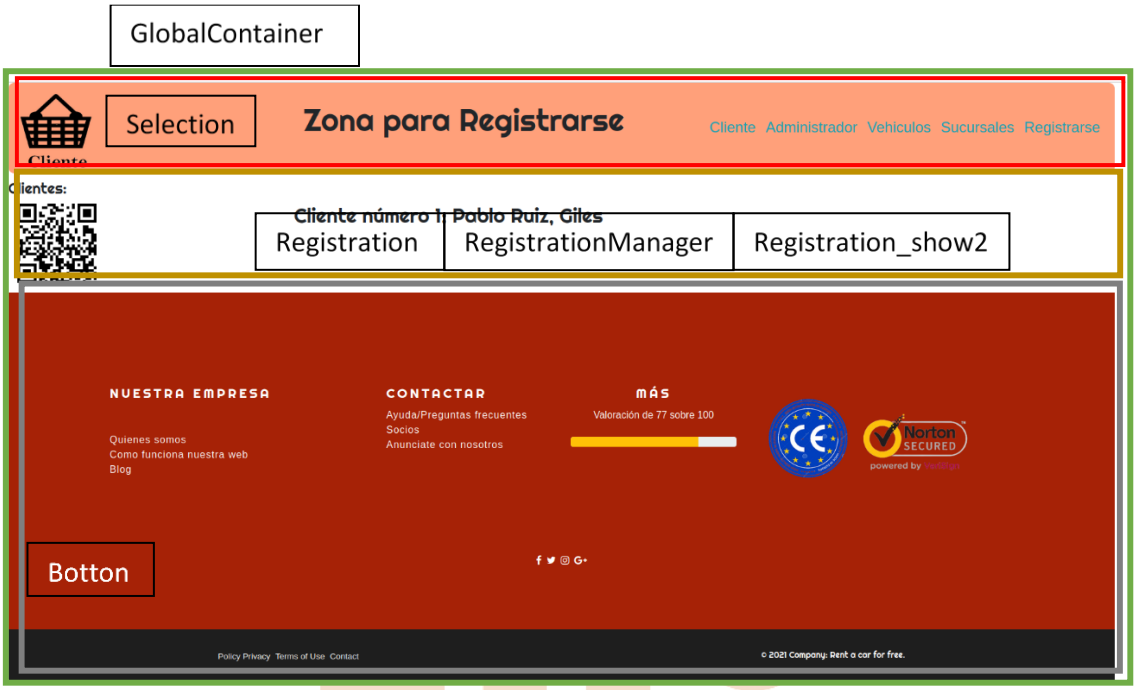

Figura 95: Página Clientes Registrados con los componentes que se compone

Estas son todas las páginas de las que está compuesto nuestro portal web, objeto del presente proyecto. Una vez descritos todos los contratos y el *frond-end*, utilizando como medio las imágenes que se acaban de mostrar o bien a través de código (como se ha hecho al principio del capítulo), solo queda demostrar su funcionamiento.

CAPÍTULO 8

Pruebas de funcionamiento

8.1. Introducción

En este capítulo se va a explicar el funcionamiento del proyecto desarrollado, indicando todas las posibles interacciones que se pueden realizar para comprobar su correcto funcionamiento, conocer las limitaciones que tiene el sistema actual según las condiciones en las que se desenvuelve, y detectar aspectos de mejora que podrán ser incluidos en futuras versiones del proyecto.

8.2. Consideraciones previas

Es necesario reseñar las limitaciones que tienen las Blockchain de prueba como *Ganache*, que ha sido la seleccionada este caso para realizar la simulación de este sistema, ya que presenta la particularidad de que lo construido y desplegado en ella es volátil. Esto nos impide guardar datos de forma indefinida. Por tanto, las cuentas que se creen de administración, clientes y demás datos, solamente serán funcionales mientras la Blockchain que creemos se mantenga desplegada y operativa. Es necesario hacer hincapié en esto porque, a continuación, se va a mostrar su funcionamiento desde el punto de vista de un administrador y desde un cliente.

8.3. Portal Web

Por defecto, los usuarios que accedan por primera vez a nuestra Web de alquiler de vehículos encontrarán las pestañas de *Cliente* y *Vehículos*, que se mostrarán de la siguiente manera:

8.3.1. Página de Clientes

Figura 96: Página Clientes por defecto

Desde esta pestaña del portal web se podrán ver los vehículos que en ese momento tenga alquilados el cliente que está *logeado* con su cuenta. También podrá, desde esta misma pestaña, devolver los vehículos si así lo desea.

8.3.2. Página de vehículos:

Figura 97: Página Vehículos por defecto

En esta pestaña los clientes sin registrar verán el mensaje “Necesita registrarse primero”. Los clientes que estén registrados en el sistema, podrán observar todos los países donde opera la empresa, y comenzar con el proceso de búsqueda del vehículo que más le guste para proceder a alquilarlo.

8.4. Funcionamiento.

8.4.1. Administrador:

Los siguientes pasos, que vamos a desarrollar, los podrán realizar los actores con el rango de administrador. Las acciones que pueden seleccionar son:

- Creación de Sucursales
- Creación de vehículos.
- Observar clientes registrados

Para una mayor optimización del sistema, las cuentas de administrador deberán ser de un número reducido, para poder realizar así un mejor control y seguimiento de los datos que son generados por estas.

Creación de sucursales.

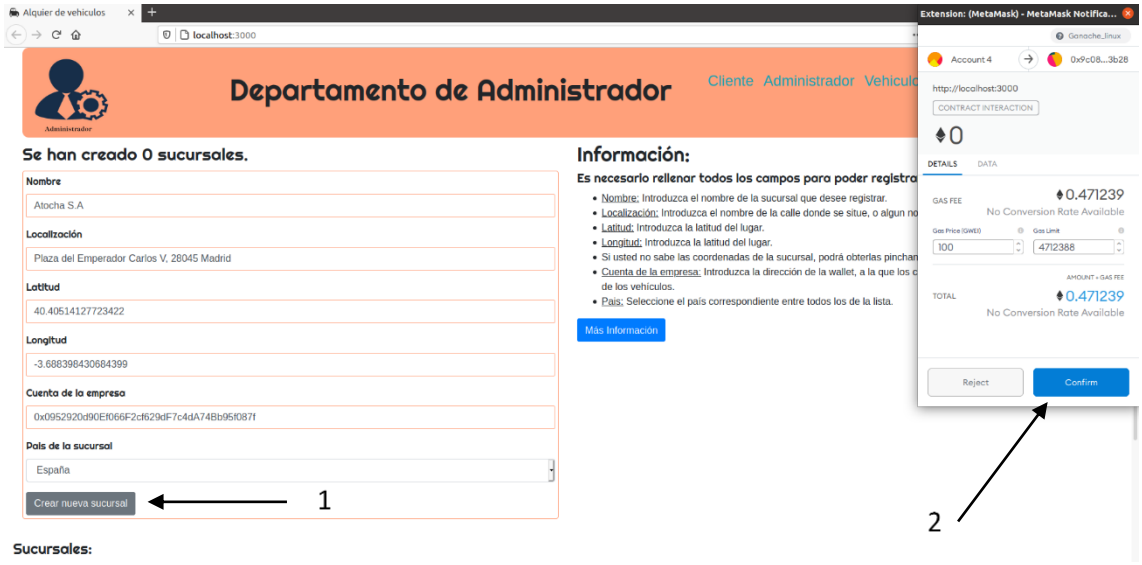

Figura 98: Creación de una nueva sucursal

Si se selecciona la pestaña de Sucursales, a continuación, se solicitará cumplimentar los campos del formulario de forma correcta para poder crear la sucursal. Hay que recordar que se han de verificar todos los campos antes de crearla, debido a que no es posible eliminarla.

Una vez rellenado todos los campos se hace click sobre “Crear nueva sucursal” (1), lo que hará que MetaMask pida conformar la acción (2) que generará una nueva sucursal en nuestra Blockchain.

Se puede ver cómo se genera de forma satisfactoria la sucursal **Atocha S.A** en el país España.

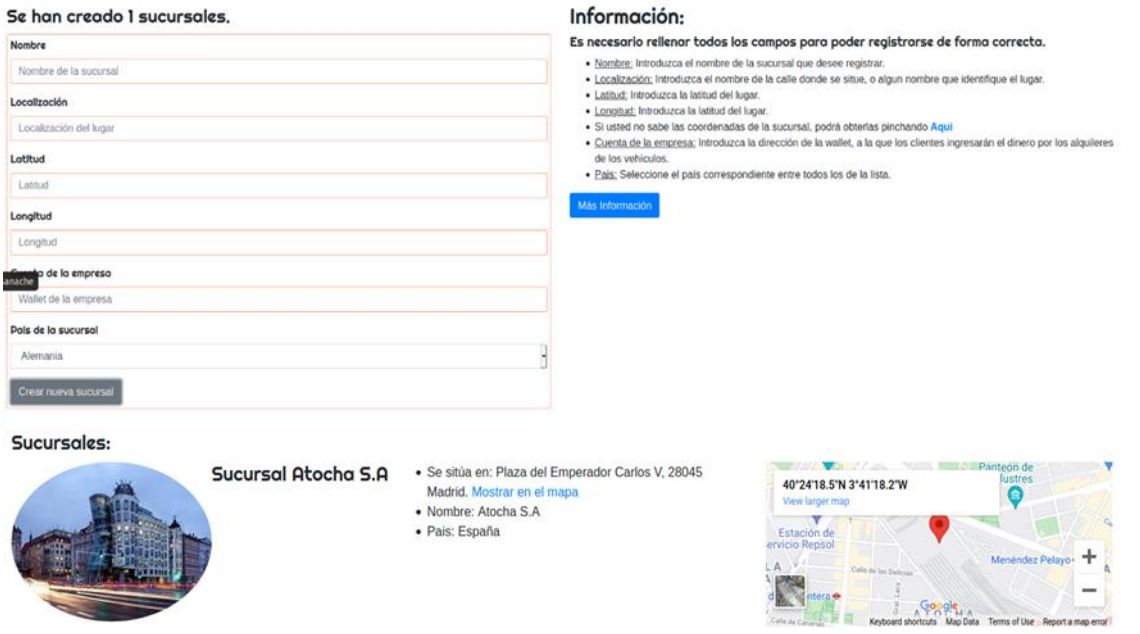

Figura 99: Visualización de la nueva sucursal

Procedemos a crear varias sucursales más para comprobar que se pueden generar en distintos países, en este caso **Alemania** y **Chipre**.

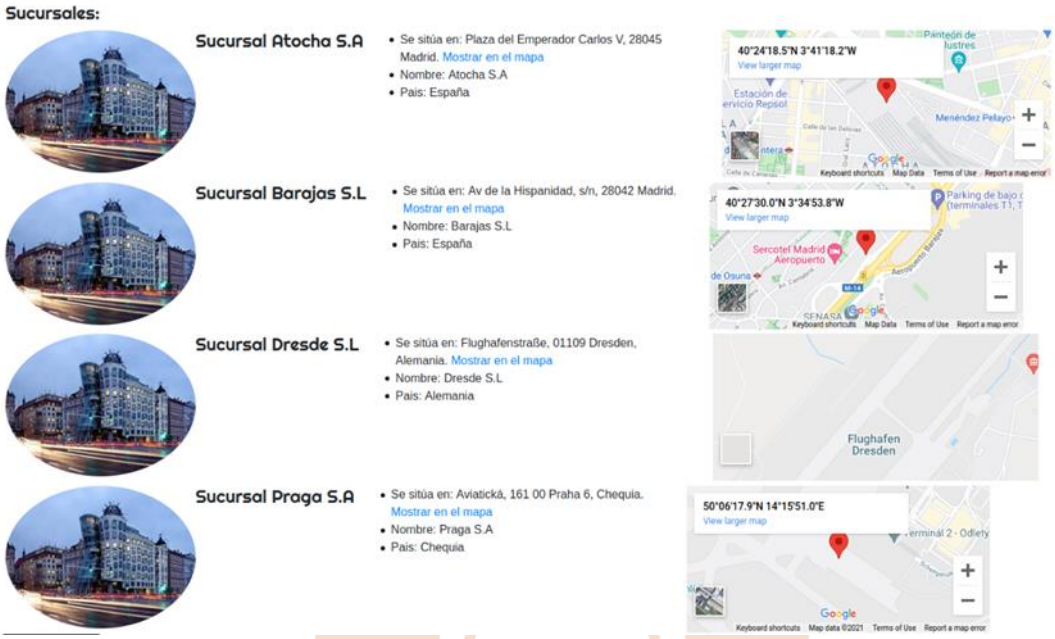

Figura 100: Visualización todas las sucursales creadas

8.4.1.1. Creación de vehículos.

Una vez que ya tenemos las sucursales, será necesario crear los vehículos de cada una de ellas.

En la pestaña de sucursales, podemos ver todas las sucursales que se han creado hasta el momento.

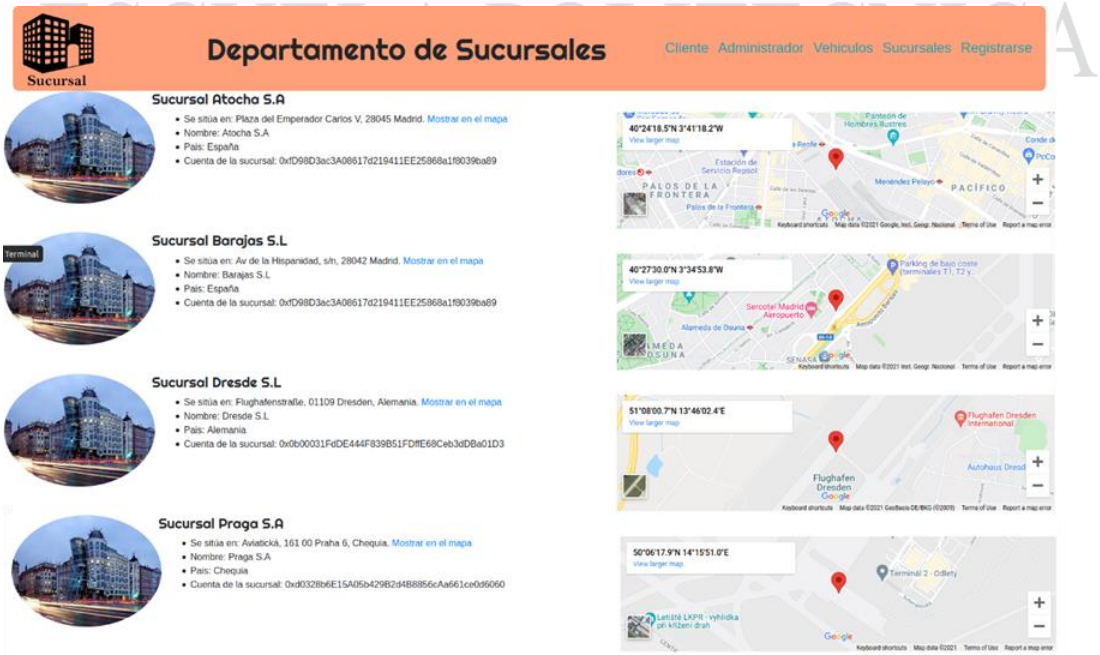

Figura 101: Pestaña Sucursales y visualización de éstas

Seleccionamos la Sucursal **Atocha S.A** y hacemos clic sobre ella. Una vez hecho esto, nos aparecerá un desplegable que deberemos rellenar de la misma forma que para la creación de sucursales.

Una vez cumplimentados todos los campos, se pulsa en “Crear nuevo vehículo” (1) y hará que aparezca la extensión MetaMask. A continuación se pulsará en “Confirm” (2). Entonces se creará el vehículo en nuestra red Blockchain.

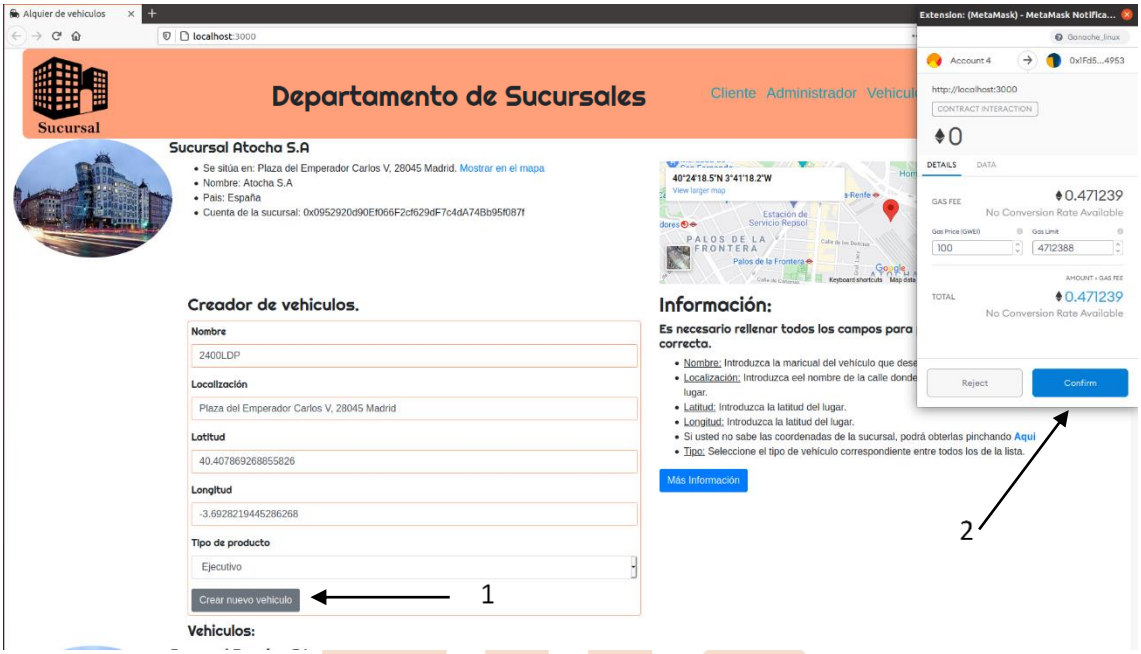

Figura 102: Creación de un nuevo vehículo

Se puede visualizar el vehículo creado en forma de código QR como en la siguiente imagen.

Forma QR:

Figura 103: Visualización del vehículo modo QR

O por el contrario, se puede visualizar toda la información disponible sobre vehículo. Se modifica la forma de visualización haciendo clic sobre el vehículo.

Forma extendida:

Figura 104: Visualización del vehículo modo extendido

Para poder mostrar el resto de los tipos de vehículos, se crearán en las diferentes sucursales los distintos tipos siguiendo los pasos descritos anteriormente.

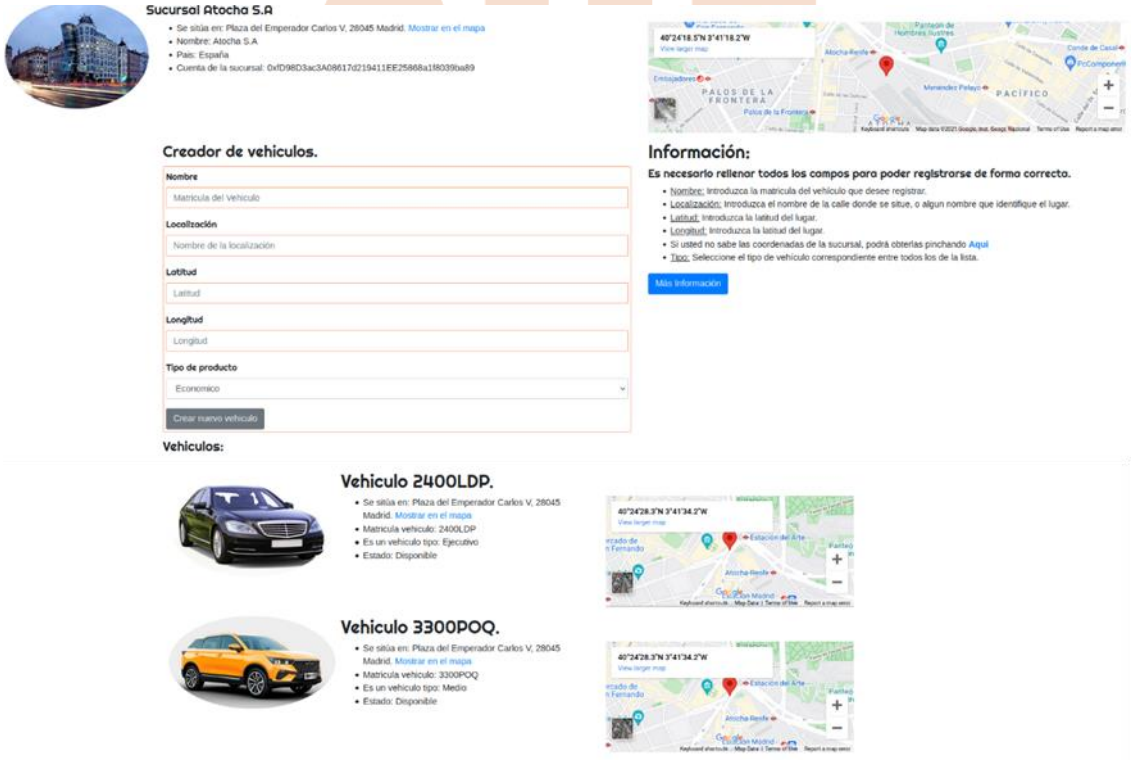

Figura 105: Visualización de los vehículos de la sucursal.

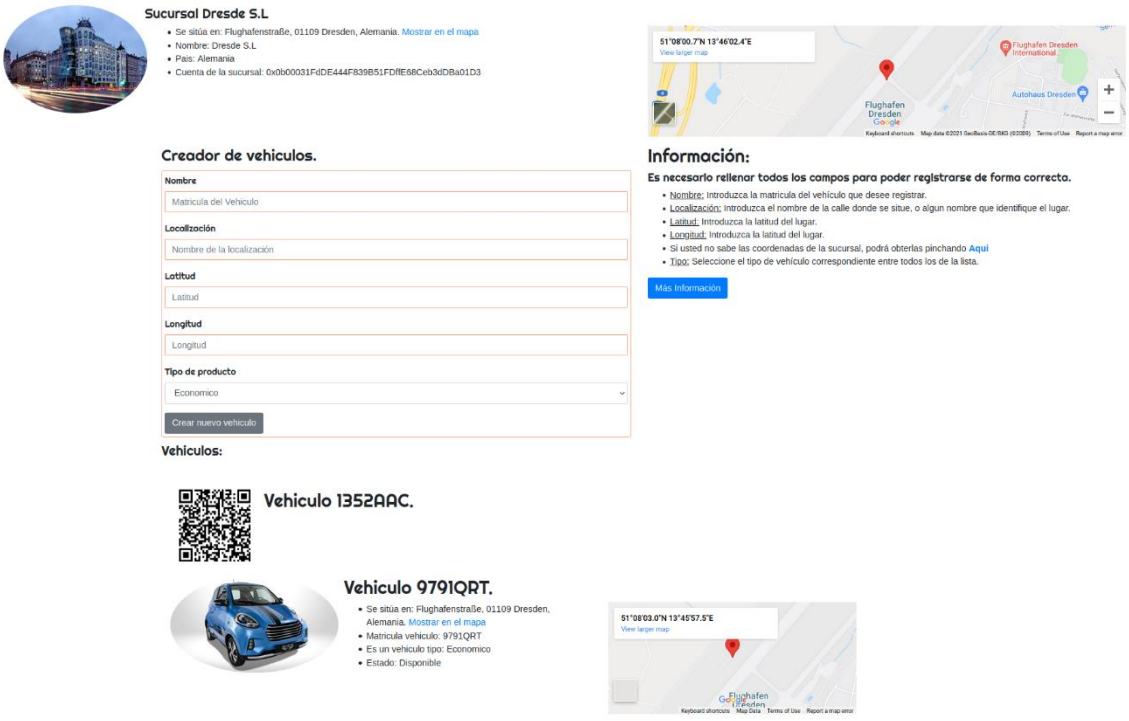

Figura 106a: Visualización de todos los vehículos del sistema

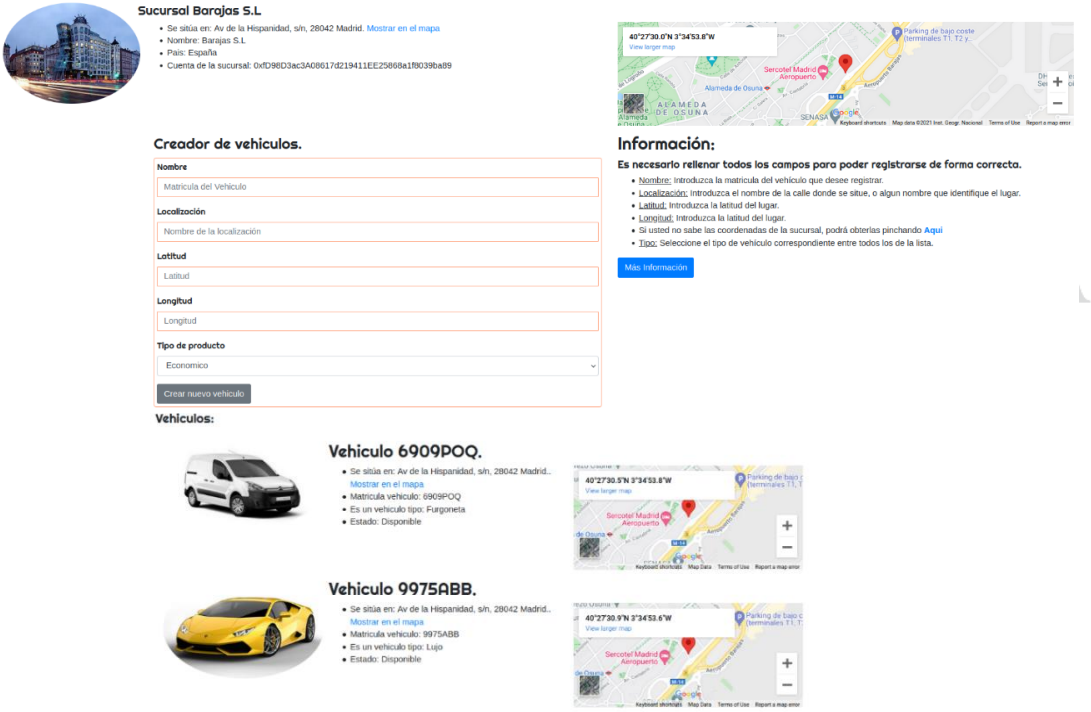

Figura 106b: Visualización de todos los vehículos del sistema

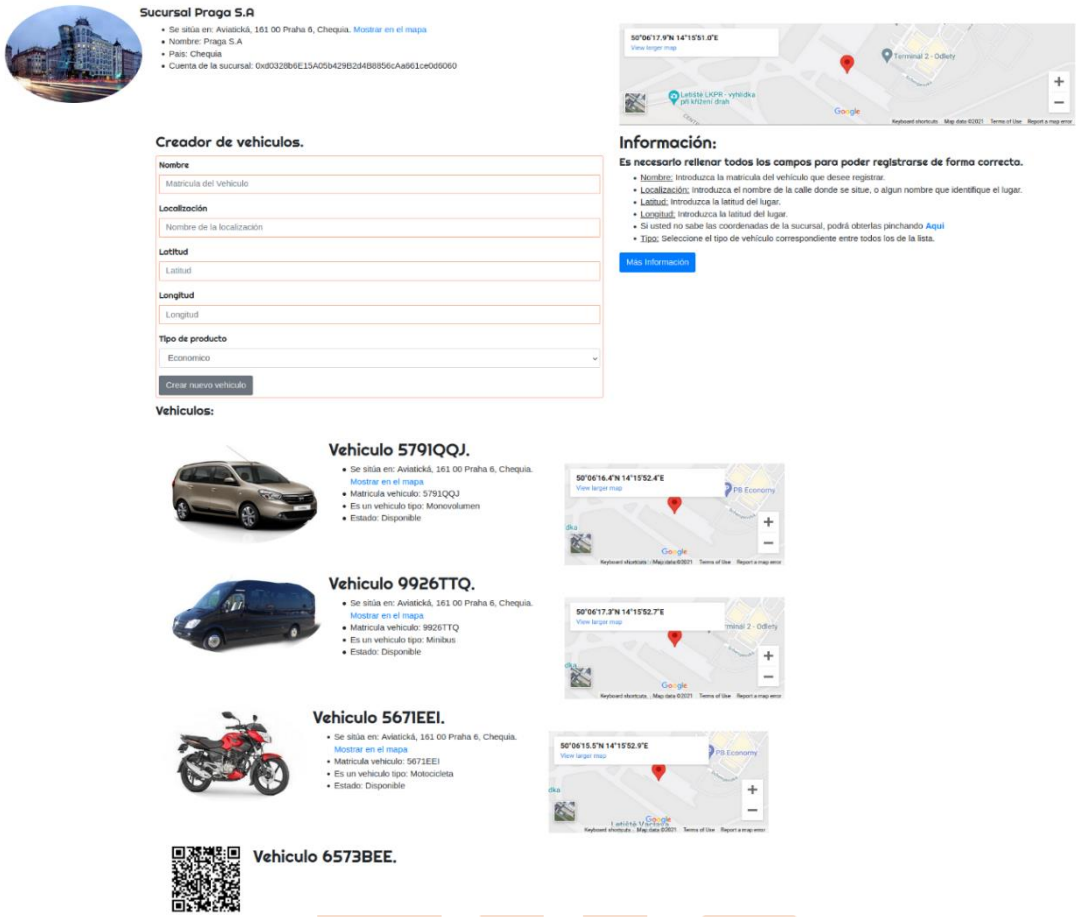

Figura 106c: Visualización de todos los vehículos del sistema

8.4.1.2. Observar clientes registrados.

Una vez creados por parte de los administradores todos los vehículos en las respectivas sucursales, la única opción permitida que resta por realizar es la de visualización de clientes registrados en el sistema, como se muestra a continuación.

Desde la pestaña de administración, se selecciona *clientes registrados* y se observa lo siguiente.

Figura 107: Base de datos de los clientes

Se muestra únicamente que hay un cliente registrado en el sistema. En este caso es el cliente que utilizaremos a continuación para mostrar el funcionamiento de nuestra Web.

Como en el anterior caso de los vehículos, se pueden visualizar los clientes por medio de un QR, que indica la dirección de la Wallet del cliente, o por medio de una lista, que indica toda la información que la Blockchain contiene sobre él.

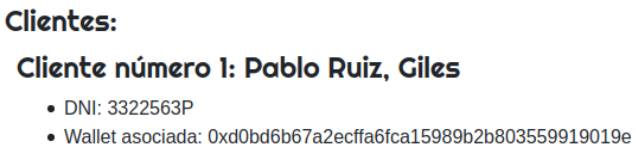

Figura 108: Visualización del cliente modo extendido

Las operaciones descritas serán las únicas que podrá realizar la figura del administrador.

8.4.2. Cliente.

Desde la perspectiva del cliente, se podrá realizar las siguientes acciones:

- Registrarse en el sistema.
- Alquilar vehículos.
- Devolver vehículos.

8.4.2.1. Registrarse en el sistema.

El cliente, por defecto, verá una página de inicio como la mostrada al principio del capítulo. Para poder alquilar vehículos, que es el objetivo de los clientes, es necesario previamente haberse registrado en el sistema. Por tanto, en esa página inicial, deberá pulsar sobre "Registrarse" y se accederá a la mostrada a continuación.

Figura 109: Proceso de registrase

Una vez cumplimentados todos los campos, se pulsará sobre "Registrarse" (1). Aparecerá la extensión Metamask y se validará en "Confirm" (2). Esto hará que se cree un nuevo usuario en nuestra red Blockchain.

Después de aceptar la transacción, indicará que se ha creado un cliente.

Figura 110: Visualización del número total de clientes registrados

El cliente una vez registrado en el sistema, podrá acceder a la pestaña de vehículos y dependiendo de sus intereses, seleccionar el deseado.

8.4.2.2. Alquiler de vehículos.

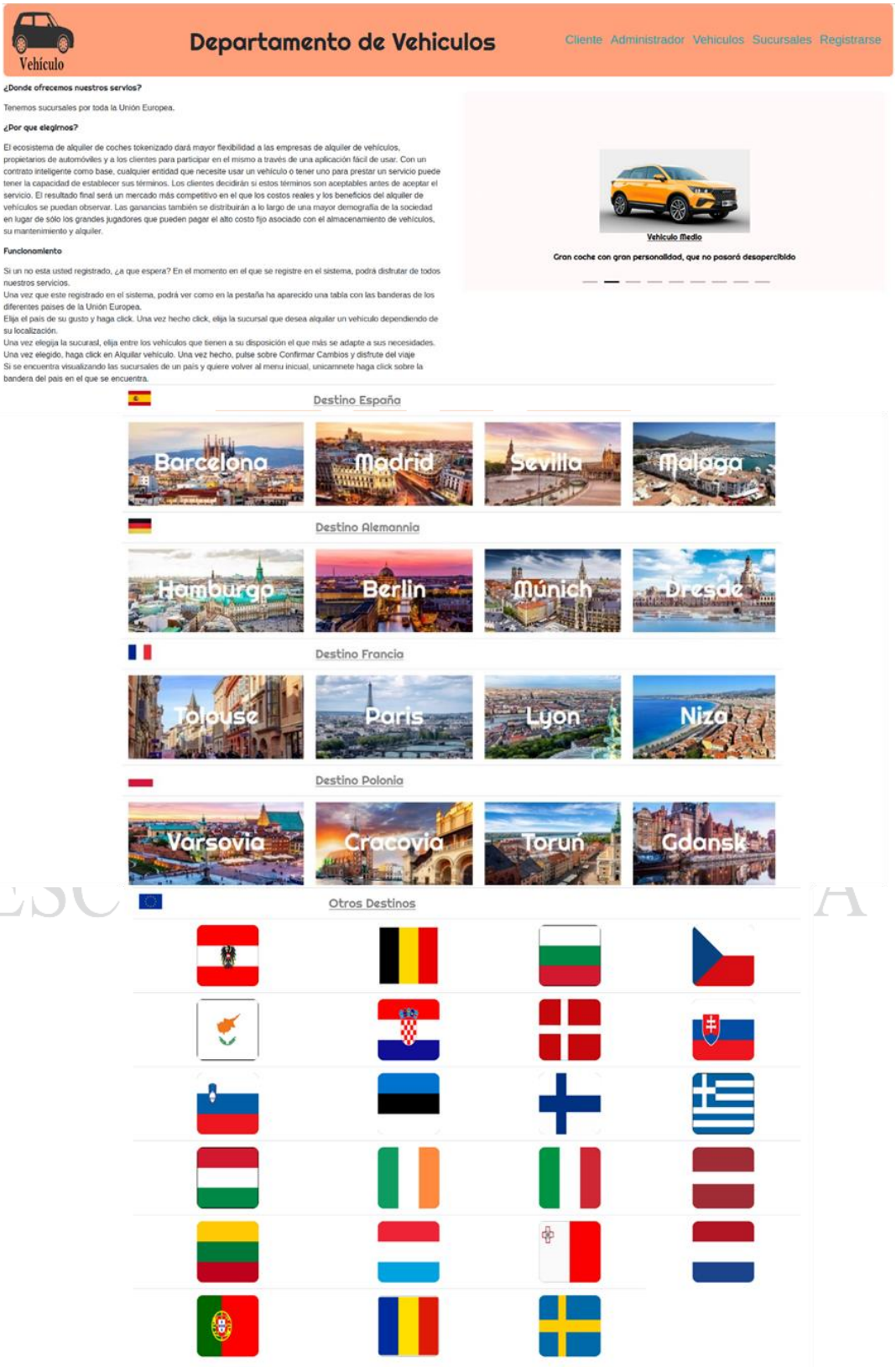

Figura 111: Visualización pestaña Vehículos con el cliente registrado

El cliente debe decidir en primer lugar, cuál es el país en el que quiere alquilar un vehículo. Para ello pulsa sobre la imagen de este.

Figura 112: Visualización sucursales país España

En este caso, se ha seleccionado el país **España**. Se pueden ver las distintas sucursales que previamente el administrador ha creado en este país.

A continuación, se selecciona la sucursal que nos interese por proximidad u otra característica, y se podrán observar los vehículos que están disponibles:

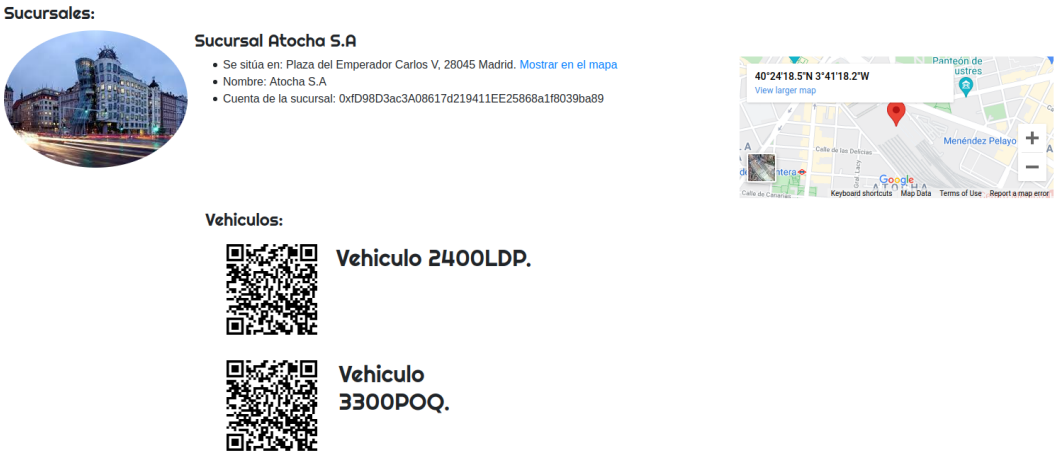

Figura 113: Visualización de los vehículos en modo QR

Por defecto, se pueden ver con el código QR para tener de esta forma toda la información más compactada.

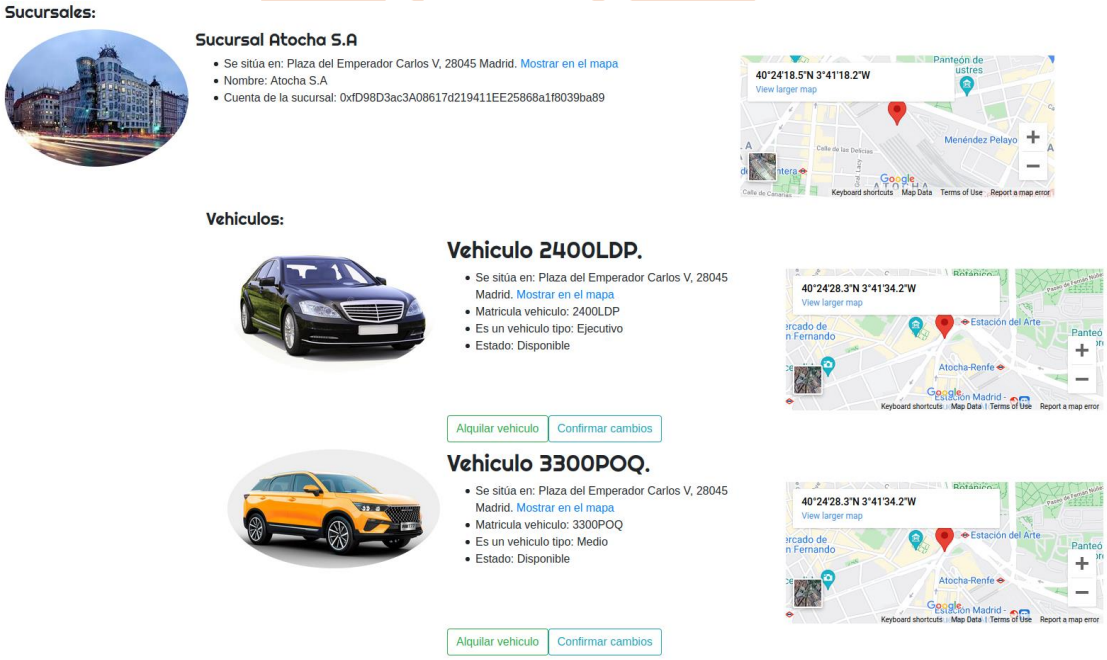

Figura 114: Visualización de los vehículos en modo extendido

Se selecciona el vehículo que se desee y se pulsa sobre “Alquilar”.

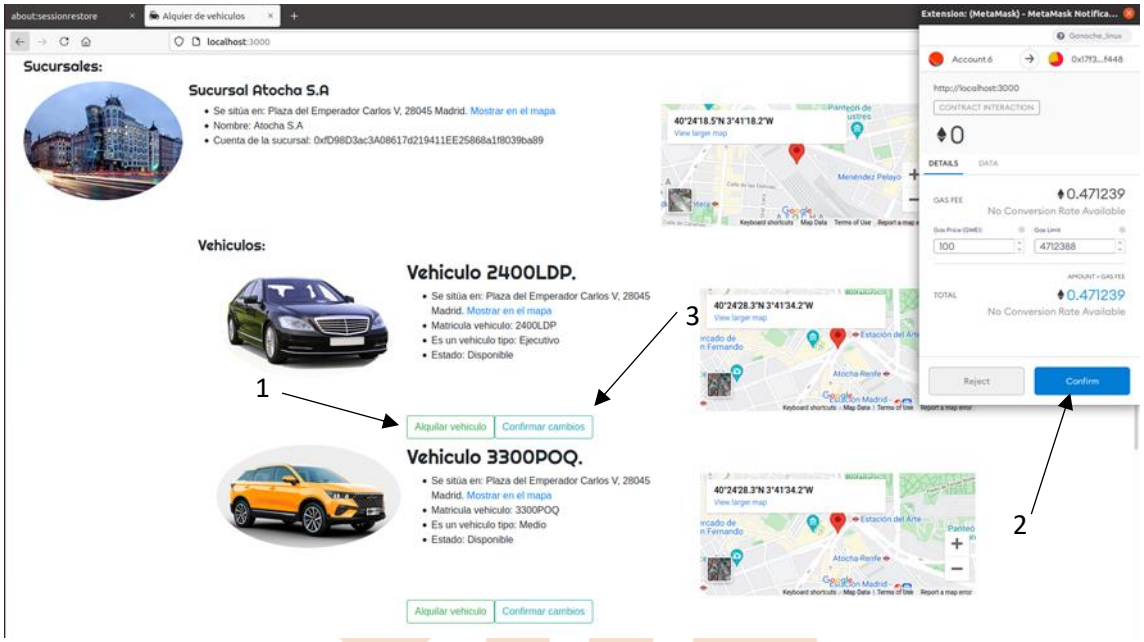

Figura 115: proceso de alquiler de vehículo

En este caso, el cliente ha seleccionado el vehículo 2400LDP. Deberá activar el botón "Alquilar". Tras pulsarlo aparecerá una ventana de MetaMask que pedirá confirmar la acción. Se deberá validar en "Confirm". Por último, solo restaría que el cliente haga clic sobre "Confirmar cambios" para poder ver en la página del "Departamento de Clientes" el vehículo que tiene actualmente en alquiler.

Comprobamos que, en la pestaña "Departamento de Clientes", se puede apreciar cómo en la cuenta "0xd0bd6b67a2eCffA6FCa15989b2B803559919019E" hay actualmente alquilado un vehículo.

Figura 116: Visualización pestaña Clientes con vehículo alquilado

Página vista desde el modo QR, o desde el modo de vista completo, en el que aparece toda la información que está disponible sobre el vehículo.

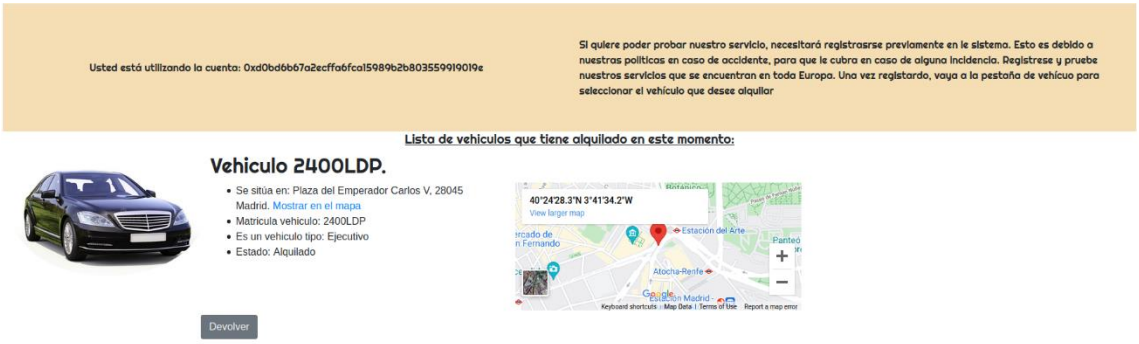

Figura 117: Visualización de la información del vehículo forma extendida

8.4.2.3. Diferenciación de usuarios.

A continuación, vamos a comprobar cómo el sistema muestra solamente los vehículos alquilados de un cliente en concreto, y que no son accesibles a cualquiera que se pueda *loguear* en nuestro portal web. Para realizar esta acción, se va a acceder a nuestra Web mediante otra cuenta. En este caso la dirección de la cuenta que vamos a utilizar es: "0x6B685fC1DbB3c9ebEef5dABAE7eB21a525B4fdfC".

Figura 118: Visualización pestaña Cliente desde otra cuenta sin vehículos alquilados

Como se puede comprobar, en el momento en el que se accede mediante otra cuenta, desaparece el vehículo que se mostraba y que está actualmente alquilado. Esto sucede porque hemos cambiado la cuenta desde la que se accede a la Web, y este usuario actualmente no tiene alquilado ningún vehículo.

Otra comprobación, que es necesaria realizar para mostrar el correcto funcionamiento del sistema, es que únicamente deben aparecer en la pestaña de "Departamento de Vehículos" los vehículos que estén disponibles, no todos los vehículos del sistema.

Para ello accedemos a la sucursal de **Atocha S.A,** y comprobamos cómo actualmente solo tiene disponible un vehículo para alquilar.

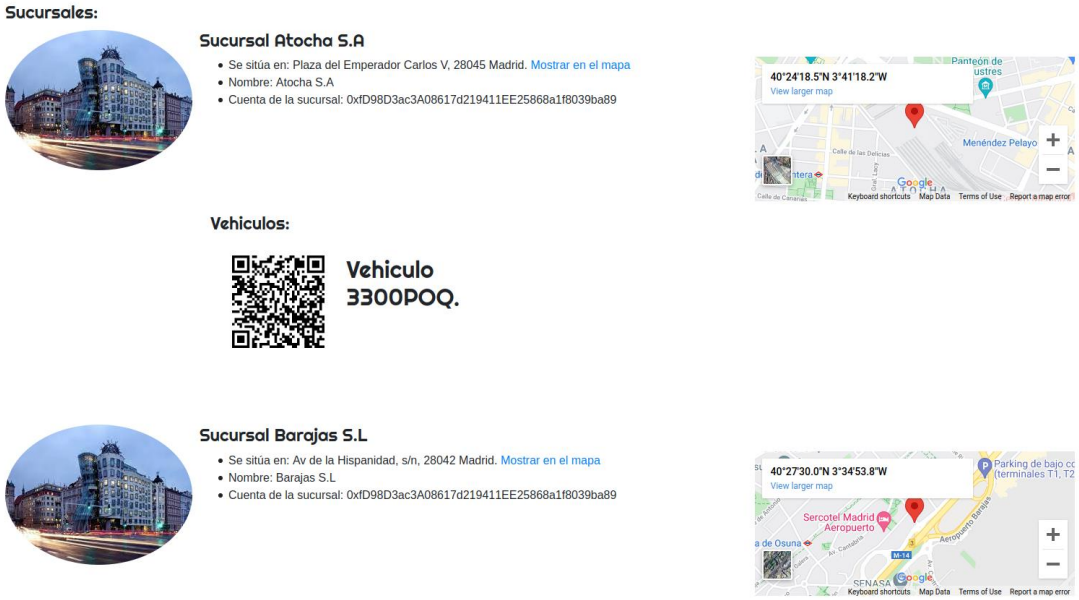

Figura 119: Visualización de los vehículos disponibles de la sucursal

Desde la pestaña *Sucursales*, se puede apreciar la lista de todos los vehículos de las distintas sucursales, tanto los vehículos alquilados como los que están disponibles. Es necesario recordar que, a esta pestaña, solo pueden acceder las cuentas de administradores. Por tanto, los clientes nunca podrán acceder a ella. Es importante reseñarlo ya que la información de los vehículos solo es accesible desde unas pestañas determinadas.

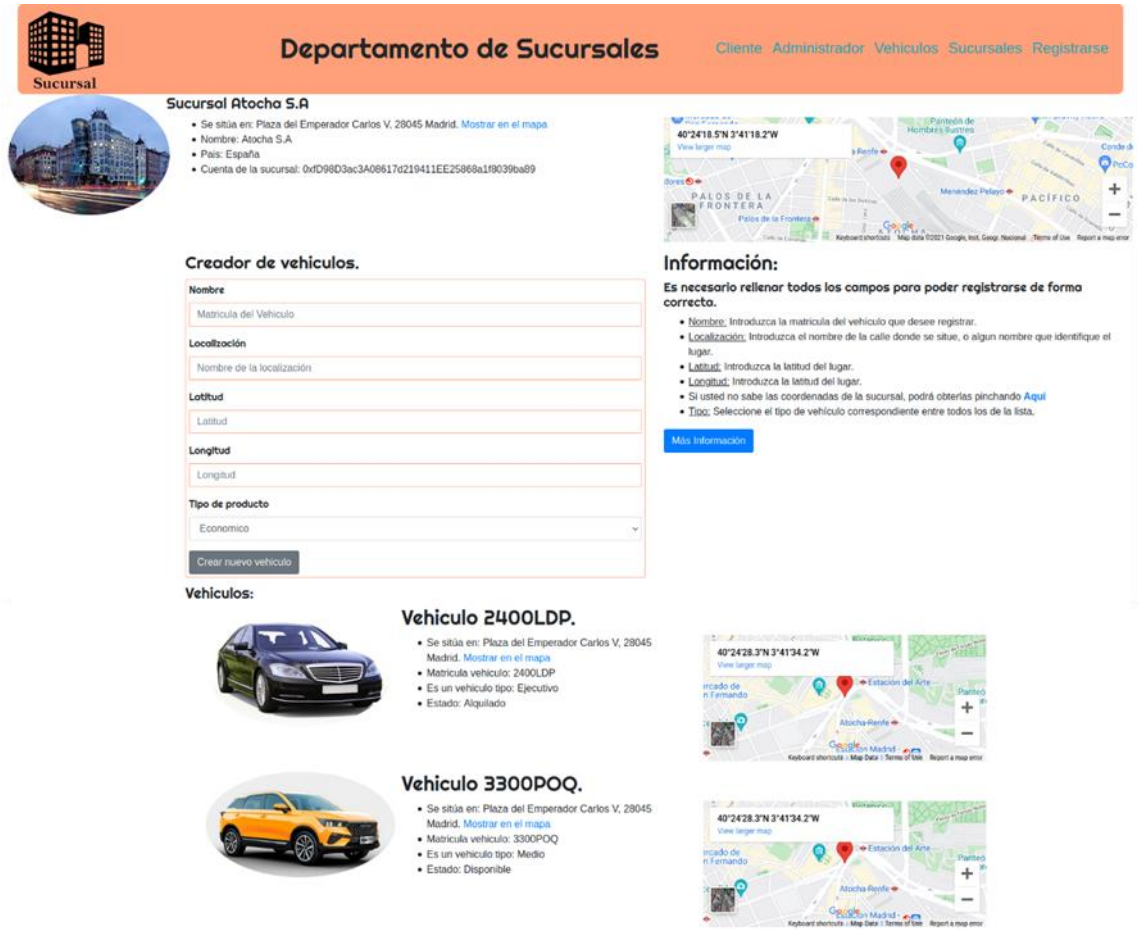

Figura 120: Visualización de los vehículos de la sucursal desde pestaña Sucursales

8.4.2.4. Devolución de vehículos.

La última comprobación por realizar será la de poder devolver el vehículo de forma satisfactoria. Para ello:

Se pulsará en el botón “Devolver” (1).

Figura 121: Proceso devolución del vehículo

Tras haber pulsado el botón “Devolver”, aparecerá el siguiente formulario que habrá que cumplimentar para poder seguir con el proceso de devolución. Una vez completado el formulario y se pulsará sobre el botón “Validar Coordenadas” (2).

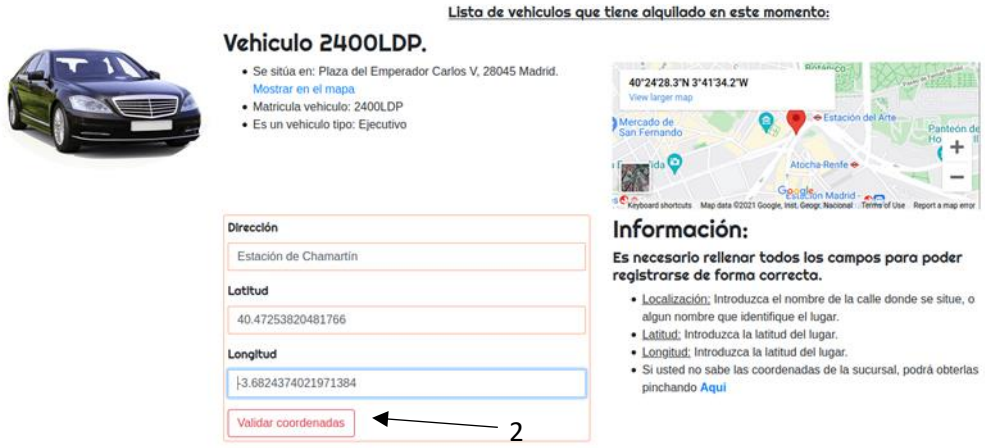

Figura 122: Proceso devolución del vehículo

Después de haberlo pulsado, aparecerá la última pestaña antes de finalizar con el proceso de devolución del vehículo. En esta se mostrará el coste del alquiler. En este caso, el que corresponda desde el día 4 al 20; es decir, 16 días. Cuyo importe final asciende a 0.224 tokens.

Figura 123: Proceso devolución del vehículo

Activamos el botón de “Pagar y devolver vehículo” (3) y aparecerá una ventana emergente de MetaMask que pedirá confirmar la transacción. Se deberá pulsar sobre “Confirm” (4). Por último, solo quedaría hacer clic sobre “Confirmar Cambios” (5) para mostrar la nueva pestaña de “Departamento de Clientes”, donde no volverá a aparecer el vehículo que se acaba de devolver.

Figura 124: Proceso devolución del vehículo

Se puede comprobar en la siguiente imagen cómo ahora la lista de vehículos es de 0 vehículos, para la cuenta "0xd0bd6b67a2eCffA6FCa15989b2B803559919019E".

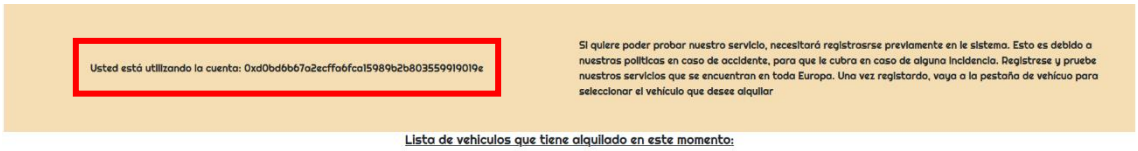

Figura 125: Comprobación del vehículo devuelto

En la siguiente imagen de *Ganache*, se puede apreciar la transacción de tokens desde la cuenta del cliente hacia la cuenta que la sucursal tiene prefijada para depositar el pago de los alquileres.

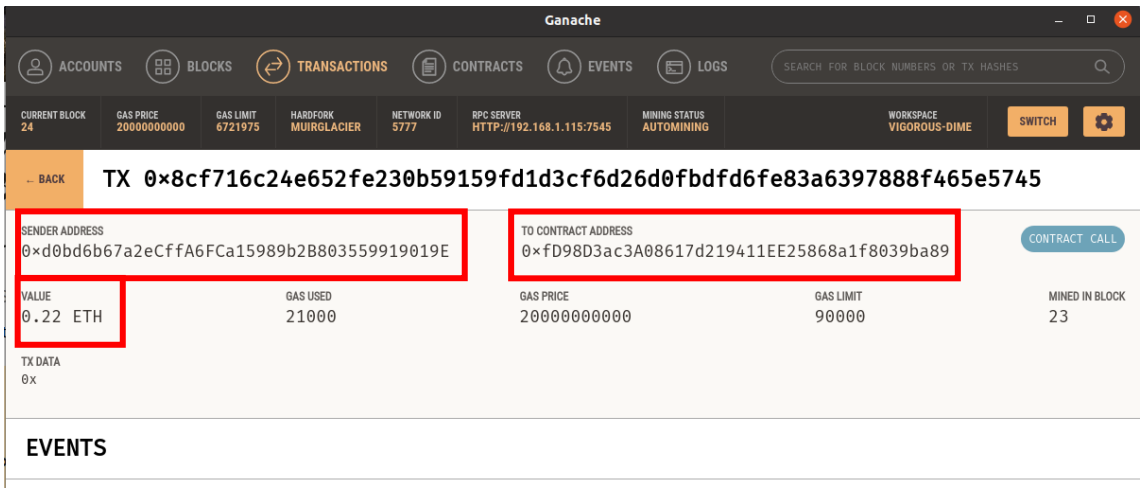

Figura 126: Comprobación de transferencia mediante Ganache

Se puede ver cómo se realiza una transferencia de 0.22 Ether desde la cuenta "0xd0bd6b67a2eCffA6FCa15989b2B803559919019E" hacia la cuenta "0xfD98D3ac3A08617d219411EE25868a1f8039ba89".

CAPÍTULO 9

Conclusiones y líneas futuras

9.1. Conclusiones.

A lo largo del desarrollo de este TFG, he podido conocer de forma más amplia las aplicaciones que tiene el uso de la tecnología Blockchain, no quedándose únicamente en el ámbito financiero.

Se puede concluir que los propósitos que se definieron al comienzo de este proyecto se han cumplido de forma satisfactoria, en especial el principal objetivo fijado, es decir, el de la integración de la tecnología Blockchain en los mercados de alquileres de vehículos.

El uso e implantación del Blockchain, en muchos de los ámbitos que utilizan distintas tecnologías sobre redes informáticas, no solo es un gran aporte sino que supone un cambio en las formas de entender y desarrollar las relaciones comerciales tal y como las conocemos actualmente. Su utilización puede evitar el uso de intermediarios, con el consiguiente abaratamiento de costes, y puede agilizar procesos que en un principio deberían de ser tediosos y costosos. Todo ello, sin perder de vista la seguridad y el asentamiento de la información distribuida que se cursa y sin posibilidad de modificación o manipulación posterior.

Por tanto, se han conseguido desarrollar todos los objetivos planteados, y realizarlo mediante el uso de herramientas software de código abierto, que me ha permitido conocer tecnologías que van a tener un amplio despliegue en un futuro cercano, dado el auge del Blockchain en la actualidad.

9.2. Líneas futuras.

En el transcurso de la elaboración de este proyecto, se han ido detectando posibles limitaciones de funcionalidad, debido en parte al tipo de tecnología Blockchain escogida en su desarrollo, así como posibles ampliaciones de características que harían este portal web más versátil. Todo ello supondría una ampliación importante de los objetivos perseguidos en este trabajo, que conllevaría una gran dedicación en tiempo y recursos, y que escapan a la finalidad del presente proyecto.

Se pueden contemplar varias posibilidades de ampliación:

- La tecnología Blockchain empleada en este trabajo es *Ethereum* y, en concreto, se ha desarrollado sobre una red de pruebas como es *Ganache*. Pero hoy día, la tecnología Blockchain está recibiendo un importante empuje por parte de grandes emporios empresariales, que soportan la creación de entornos Blockchain permisionados y con gran capacidad de gestión por parte de las empresas que los desarrollan (*TrustOS* de Telefónica, *Amazon Managed Blockchain*, *IBM Blockchain*, Alastría como red pública…), permitiendo al cliente desentenderse de los procesos de mantenimiento y gestión de la infraestructura creada.
  Esto da lugar a importantes posibilidades de ampliación del sistema que se ha desarrollado en este trabajo, tanto en el ámbito de la seguridad (registros de clientes y empresas, mayor seguridad en la tramitación de datos en el entorno Web,…), como permitiendo una mayor velocidad en la tramitación y verificación de los *Smart Contracts*. También en el entorno de la gestión, ya que estas empresas disponen de gestión centralizada que permite a las empresas explotadoras de *rent a car* desligarse de la misma, contratando estos servicios a dichas operadoras.

- Actualmente los *Smartphones* son los dispositivos con mayor grado de difusión y utilización por parte de la población. Eso implica que, si deseamos darle una mayor proyección a nuestro sistema, deberíamos crear una aplicación (*App*) del mismo para que se pueda ejecutar directamente en estos dispositivos.
  Esto en sí supone trabajar con otros lenguajes de programación y otros sistemas operativos. Por tanto, supone un esfuerzo adicional a lo ya creado en el presente proyecto.

Apéndice A
Memoria económica

A.1 Planificación y estimación económica.

Al abordar el presente proyecto, la planificación de su ejecución ha sido primordial, debido a que una correcta estructuración del trabajo determina una mayor optimización de los tiempos y, por consiguiente, un mayor rendimiento en su logro.

El desarrollo de este trabajo se ha dividido en tres fases: el estudio de la tecnología, el desarrollo y depuración de la aplicación software y la redacción de toda la documentación necesaria.

Hay que resaltar que, el hecho de utilizar herramientas *Open Source* supone que no necesitaremos ninguna licencia y, por tanto, no conllevará un coste extra a nuestra aplicación. Hoy día el uso de aplicaciones de código abierto supone una tendencia al alza, y vemos cómo instituciones y administraciones públicas apuestan por ello. Además, nos aseguraremos estar al día en las novedades y las actualizaciones que surjan.

En la siguiente tabla se muestran los costes del desarrollo y puesta en servicio de este proyecto piloto.

Concepto	Descripción	Coste unitario	Coste total
Equipo informático	Ordenador con las siguientes características: i7 6700HQ, 16 GB RAM, 1 TB SSD, GTX 990 M.	1200 €	1200 €
VMware Workstation Player.	Máquina virtual utilizada para emular el sistema operativo Linux.	0 €	0 €
Linux	Sistema operativo empleado para ejecutar el proyecto. GNU Ubuntu 20.04 LTS.	0 €	0 €
Visual Studio Code	IDE de desarrollo de Microsoft	0 €	0 €
REMIX	IDE de desarrollo de Solidity.	0 €	0 €
MetaMask	Plugin para clientes Web para operar con Wallets	0 €	0 €
Node.js	Entorno de ejecución de JavaScript	0 €	0 €
React	Biblioteca GUI de código abierto de JavaScript.	0 €	0 €
Bootstrap	Framework CSS empleado para desarrollar aplicaciones y sitios Web	0 €	0 €
Truffle Suite	Conjunto de herramientas para el desarrollo de aplicaciones de Blockchain. Se compone de: *Ganache, Truffe, Drizzle* y las *Truffles Boxes*.	0 €	0 €
Trabajo del desarrollo	Comprende las etapas de investigación, diseño del proyecto y puesta de funcionamiento. También incluye el tiempo dedicado a la memoria del proyecto. Se ha dedicado un total de 300 horas para su completo desarrollo	30 €	9000 €
Total			10200 €

Apéndice B
Manual de instalación

En este apéndice se van a explicar los pasos que se deben seguir, a fin de poder instalar todo el software que se requiere, para el correcto funcionamiento de la aplicación aquí desarrollada. Es necesario que, previamente, el usuario disponga en su equipo del sistema operativo Ubuntu 20.04 LTS.

B. 1 Instalación.

Antes que nada, hay que actualizar el sistema operativo para poder evitar problemas de versiones. Por tanto, se ejecuta en el Terminal lo siguiente:

```
pablo@ubuntu:~$ sudo apt-get update
```

Figura 127: Comando de Ubuntu

Una vez actualizado, procederemos a instalar todos los programas y librerías necesarias.

Empezaremos instalando *Ganache*. Para ello, necesitaremos descargarlo desde el siguiente enlace, como se muestra a continuación.

Enlace: https://www.trufflesuite.com/ganache

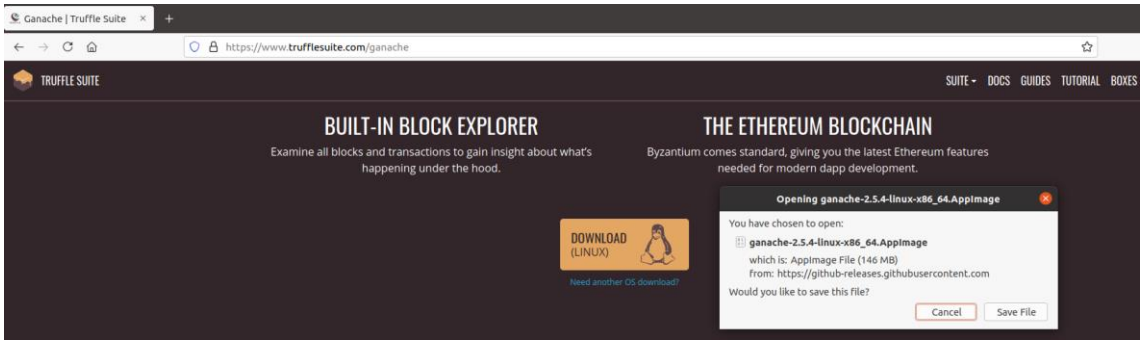

Figura 128: Descarga de Ganache

Una vez descargado, únicamente será necesario ejecutar la siguiente línea de comandos dentro de la carpeta donde se encuentre guardado el archivo que se acaba de descargar.

```
pablo@ubuntu:~/Desktop/ganache$ sudo ./ganache-2.5.4-linux-x86_64.AppImage
```

Figura 129: Comando de Ubuntu

Una vez instalado *Ganache*, vamos a proceder a descargar *Node js*. Para su descarga, será necesario escribir y ejecutar la siguiente línea en el terminal del S.O.

```
pablo@ubuntu:~$ sudo apt install nodejs
```

Figura 130: Comando de instalación de Node JS

Comprobamos que se ha instalado de forma satisfactoria mediante el siguiente comando.

```
pablo@ubuntu:~$     nodejs -v
v10.19.0
```

Figura 131: Comprobación de versión de Node JS

Se puede ver cómo se ha instalado de forma correcta. Actualmente tenemos la versión 10.19.0.

Este paso será necesario realizarlo, ya que es un requisito previo antes de pasar a instalar *Truffle*.

```
pablo@ubuntu:~$ npm install -g truffle
```

Figura 132: Comando de Ubuntu

Gracias a incorporar el modificador "-g", *Truffle* se instalará globalmente en el sistema operativo, no quedando sujeto a la carpeta desde donde se está ejecutando la línea de comandos.

Tras haber instalado tanto *Ganache* como *Node.js* y *Truffle*, únicamente nos queda por descargar las *truffle boxes* y las librerías extras que se utilizan en el proyecto.

Lo siguiente a realizar, será descargar el proyecto de *petshop.*

```
pablo@ubuntu:~/Desktop/TFG$ truffle unbox pet-shop
```

Figura 133: Comando de tRUFFLE

Si se ha descargado correctamente, deberá aparecer el siguiente mensaje donde se puede leer: “Unbox successfull, sweet!”.

```
Starting unbox...
==================

✔ Preparing to download box
✔ Downloading
npm WARN pet-shop@1.0.0 No description
npm WARN pet-shop@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules/fsevents)
:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4:
 wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

✔ Cleaning up temporary files
✔ Setting up box

Unbox successful, sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test
  Run dev server: npm run dev
```

Figura 134: Comprobación de descarga exitosa

Una vez descargado, procederemos a instalar lo siguiente:

```
pablo@ubuntu:~/Desktop/TFG$ npm install -g create-react-app
```

Figura 135: Comando de Node JS

Es necesario realizar este paso debido a que nos permitirá crear una carpeta con todo el contenido necesario para poder ejecutar el portal web. Una vez instalado, al igual que se hizo en el caso de *Truffle*, procederemos a crear una carpeta con el nombre “public”, que contendrá todo lo relacionado con el entorno Web.

```
Creating a new React app in /home/pablo/Desktop/TFG/public.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

Figura 136: Instalación de programa

Cuando tengamos todo descargado, solo nos restará comprobar los paquetes que están instalados en el proyecto. Los que falten se instalarán de forma manual.

Ejecutando el siguiente comando, se pueden comprobar los paquetes que actualmente se encuentran instalados en el proyecto.

```
pablo@ubuntu:~/Desktop/TFG/public$ npm list
```

Figura 137: Comando de Node JS

Para realizar la instalación de todos los paquetes a fin de que el sistema funcione de forma correcta, será necesario ejecutar el siguiente comando donde únicamente se sustituirá "nombre_del_paquete" por el nombre del paquete a instalar.

```
pablo@ubuntu:~/Desktop/TFG/public$ npm install nombre_del_paquete
```

Figura 138: Comando de Node JS

Se debe comprobar que la lista final de paquetes instalados es idéntica a la expuesta a continuación.

- @drizzle/React-components
- @drizzle/React-plugin
- @drizzle/store
- bootstrap
- drizzle
- Drizzle-react
- jqery
- Qrcode.react
- react
- React-bootstrap-carousel
- React-dom
- React-scripts
- reactstrap
- serve
- Web3

Si todo es correcto, tendremos instalado todo el software necesario para poder ejecutar nuestra aplicación Web.

Apéndice C
Bibliografía

C.1 Referencias.

[1] «Desarrollo activo,» [En línea]. Available: https://desarrolloactivo.com/blog/centralized-decentralized-distributed-p2p/. [Último acceso: 17 Febrero 2021].

[2] «Desde linux,» [En línea]. Available: https://blog.desdelinux.net/descentralizar-internet-redes-descentralizadas-servidores-autonomos/. [Último acceso: 17 Febrero 2021].

[3] «Ethereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/nodes-anc clients/. [Último acceso: 22 Marzo 2021].

[4] M. STAMP, Information security: principles and practice, John Wiley & Sons, 2011.

[5] «Hard Zone,» [En línea]. Available: https://hardzone.es/noticias/tarjetas-graficas/mineri via-gpu-motivos/. [Último acceso: 27 Marzo 2021].

[6] «Ethereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/consensus mechanisms/pos/#:~:text=Pros%20and%20cons%20%20%20%20Pros%20,secure%20sha ng.%20Shar%20...%20%20%20%20. [Último acceso: 27 Marzo 2021].

[7] «ABANCA innova,» [En línea]. Available: http://abancainnova.com/es/opinion/los-tipos-

blockchain-publica-privada-o-consorcio-explicados/. [Último acceso: 2 Abril 2021].

[8] «Avance Digital,» [En línea]. Available: https://avancedigital.mineco.gob.es/es-ES/Participacion/RespuestasEstatregiaDigital/asociacion-alastria.pdf#:~:text=Alastria%20es%20un%20consorcio%20multisectorial%20sin%20%C3%A1nimo%20de,de%20una%20red%20semip%C3%BAblica%20permisionada%20Blockcha%20%28. [Último acceso: 2 Abril 2021].

[9] «Cmc markets,» [En línea]. Available: https://www.cmcmarkets.com/es-es/aprenda-a-operar-con-criptomonedas/que-es-ethereum#:~:text=Ethereum%20es%20una%20plataforma%20digital,al%20mercado%20%20las%20criptomonedas. [Último acceso: 8 Abril 2021].

[10] «bitnovo Blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entı ethereum-y-ethereum-classic/. [Último acceso: 9 Abril 2021].

[11] «bit2me academy,» [En línea]. Available: https://academy.bit2me.com/que-es-ethereun classic-etc-criptomoneda/. [Último acceso: 9 Abril 2021].

[12] «Blockchain Council,» [En línea]. Available: https://www.blockchain-council.org/blockchain/a-quick-guide-to-ethereum-casper/#:~:text=Ethereum%20Casper%20is%20the%20POS%20%28Proof-of-Stake%29%20protocol%20that,two%20co-developed%20Casper%20implementations%20in%20the%20Ethereum%20ecosystem.. [Último acceso: 13 Abril 2021].

[13] «Ethreum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].

[14] «Ethereum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].

[15] «Miethereum,» [En línea]. Available: https://www.miethereum.com/ether/gas/#toc1. [Último acceso: 15 Abril 2021].

[16] «Blockchain Economía,» [En línea]. Available: https://www.blockchaineconomia.es/walle blockchain/. [Último acceso: 16 Abril 2021].

[17] «Criptonoticias,» [En línea]. Available: https://www.criptonoticias.com/aplicaciones/metamask-puente-ethereum-navegador/. [Último acceso: 20 Abril 2021].

[18] «nodejs,» [En línea]. Available: https://nodejs.org/es/about/. [Último acceso: 20 Abril 2021].

[19] «TRUFFLE SUITE,» [En línea]. Available: https://www.trufflesuite.com/drizzle. [Último acceso: 24 Abril 2021].

[20] «CRYPTO FOX,» [En línea]. Available: https://crypto-fox.ru/faq/v-chem-sut-protokola-casper/. [Último acceso: 10 Abril 2021].

[21] «Mi Ethereum,» [En línea]. Available: https://www.miethereum.com/ether/gas/#toc1. [Último acceso: 18 Abril 2021].

[22] «Mi Ethereum,» [En línea]. Available: https://www.miethereum.com/ether/bitcoin-vs-ethereum/#toc13. [Último acceso: 8 Abril 2021].

[23] «bitnovo blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entı ethereum-y-ethereum-classic/. [Último acceso: 8 Abril 2021].

[24] «Ethereum Homestead,» [En línea]. Available: https://ethdocs.org/en/latest/ether.html. [Último acceso: 15 Abril 2021].

[25] «Github,» [En línea]. Available: https://github.com/uberVU/react-guide/blob/master/props-vs-state.md. [Último acceso: 20 Abril 2021].

[26] «Ethrereum,» [En línea]. Available: https://ethereum.org/en/developers/docs/. [Último acceso: 24 Abril 2021].

[27] «DigiByte HISPANO,» [En línea]. Available: https://www.digibytehispano.org/como-funciona-una-red-descentralizada/. [Último acceso: 18 Febrero 2021].

[28] «SECURED COIN,» [En línea]. Available: https://invertirencriptomonedas.es/que-es-una-criptomoneda [Último acceso: 17 Febrero 2021].

[29] «GENBETA,» [En línea]. Available: https://www.genbeta.com/desarrollo/que-son-y-para que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales. [Último acceso: 22 Febrer 2021].

[30] «bit2me,» [En línea]. Available: https://explorer.bit2me.com/btc/block/00000000839a8e6886ab5951d76f411475428afc 947ee320161bbf18eb6048. [Último acceso: 22 Marzo 2021].

[31] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle#/media/Archivo:Hash_Tree.svg. [Último acceso: 27 Marzo 2021].

[32] N. Rodriguez, «101 Blockchains,» [En línea]. Available: https://101blockchains.com/es/algoritmos-de-consenso-blockchain/#1. [Último acceso: Abril 2021].

[33] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Ethereum#/media/Archivo:ETHEREUM-YOUTUBE-PROFILE PIC.png. [Último acceso: 8 Abril 2021].

[34] S. Lauri, «bitnovo Blog,» [En línea]. Available: https://blog.bitnovo.com/cual-es-la-diferencia-entre-ethereum-y-ethereum-classic/. [Último acceso: 9 Abril 2021].

[35] «cibercafe,» [En línea]. Available: http://modvsub1-2.blogspot.com/2016/05/vmware-workstation-pro.html. [Último acceso: 20 Abril 2021].

[36] «DEV,» [En línea]. Available: https://dev.to/glaucia86/14-tips-para-optimizar-su-visual-studio-code-40ia. [Último acceso: 20 Abril 2021].

[37] «Github,» [En línea]. Available: https://github.com/ethereum/remix-ide/releases. [Últim acceso: 21 Abril 2021].

[38] «Metamask,» [En línea]. Available: https://metamask-devcon4.herokuapp.com/#slide=1 [Último acceso: 22 Abril 2021].

[39] «Node js,» [En línea]. Available: https://download.logo.wine/logo/Node.js/Node.js-Logo.wine.png. [Último acceso: 22 Abril 2021].

[40] N. Birch, «onexxtrapixel,» [En línea]. Available: https://onextrapixel.com/increase-knowledge-of-react-js-with-these-helpful-resources/. [Último acceso: 23 Abril 2021].

[41] «freepikpsd,» [En línea]. Available: https://freepikpsd.com/bootstrap-icon-png/314535/ [Último acceso: 23 Abril 2021].

[42] «Truffle Suite,» [En línea]. Available: https://www.trufflesuite.com/docs/ganache/overview. [Último acceso: 24 Abril 2021].

[43] «Github,» [En línea]. Available: https://github.com/trufflesuite. [Último acceso: 24 Abril 2021].

[44] «npm,» [En línea]. Available: https://www.npmjs.com/package/@drizzle/store. [Último acceso: 24 Abril 2021].

[45] «Truffle Suite,» [En línea]. Available: https://www.trufflesuite.com/drizzle. [Último acces 24 Abril 2021].

[46] «ANQUES TECNOLABS,» [En línea]. Available: https://www.anques.com/top-solidity-development-company/. [Último acceso: 24 Abril 2021].

[47] F. Canós, «dairioabierto,» [En línea]. Available: https://www.diarioabierto.es/378175/los-bloques-del-blockchain [Último acceso: 22 Marzo 2021].

[48] «101blockchain,» [En línea]. Available: https://101blockchains.com/es/empresas-implementando-blockchain/. [Último acceso: 1 Julio 2021].

[49] «criptomoneda.ninja,» [En línea]. Available: https://criptomoneda.ninja/blockchain-ventajas-desventajas/#blockchain-ventajas-desventajas. [Último acceso: 1 Julio 2021].

[50] «101blockchains,» [En línea]. Available: https://101blockchains.com/es/historia-de-la-blockchain/. [Último acceso: 1 Julio 2021].

Universidad de Alcalá