Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

# Trabajo Fin de Máster

## Aplicación de Blockchain para el uso de transportes

*Blockchain application for transport use*

Juan Pablo Claros Romero

Máster Universitario en Ciberseguridad e Inteligencia de Datos

La Laguna, 29 de junio de 2020

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

D. **Néstor García Moreno**, con NIF 79.085.553-F contratado por proyecto de investigación adscrito al Departamento de Ingeniería Informática y de Sistemas, como cotutor

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

Que la presente memoria titulada:

"*Aplicación de Blockchain para el uso de transportes*"

ha sido realizada bajo su dirección por D. **Juan Pablo Claros Romero**, con NIF 42.279.940-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 29 de junio de 2020

# Agradecimientos

Me gustaría agradecer a mi tutora Pino Caballero Gil, y a mi cotutor Néstor García Moreno, por su implicación y ayuda durante el desarrollo del trabajo realizado.

También me gustaría agradecer a mi familia por su continuo apoyo, y a todos mis amigos y las personas que han estado conmigo desde que empecé a trabajar en el proyecto.

# Licencia

**Resumen**

*La aparición de la tecnología Blockchain y los contratos inteligentes*[1] *ha supuesto un gran avance tecnológico ya que, en general, implican la adición de seguridad, control y confianza en la mayoría de los sectores en los que se utiliza. Dicha tecnología ha pasado de ser una absoluta desconocida a convertirse en el foco de la innovación de varios sectores (mayoritariamente el financiero), ya que permite realizar transacciones entre participantes de forma segura, confiable e irreversible. Uno de sus puntos fuertes es que no requiere la intervención de terceros para establecer relaciones de confianza entre las partes. Esto es posible gracias a que la información está distribuida en diferentes nodos que son independientes entre sí, pues la registran y la validan sin necesidad de que haya confianza entre ellos. Una de las grandes ventajas que aporta la tecnología Blockchain es que los datos de las transacciones una vez estén registradas, son imposibles de falsificar.*

*Uno de los sectores en los que más aplicabilidad tiene esta tecnología es el transporte. Este Trabajo Fin de Máster está enfocado a realizar una aplicación web descentralizada en la plataforma de Ethereum, que genere una ruta o recorrido,incluyendo diferentes medios de transporte para trasladarse de un punto A a otro punto B.*

**Palabras clave:** Blockchain, contrato inteligente, Ethereum, seguridad.

[1] Programas informáticos que se programan para que ejecuten de forma automática acuerdos que hayan determinado dos o más partes.

**Abstract**

*The appearance of Blockchain technology and smart contracts [2] has been a great technological advance since, in general, they involve the addition of security, control and trust in most of the sectors in which it is used. This technology has gone from being an absolute unknown to becoming the focus of innovation in several sectors (mainly the financial sector), because it allows transactions between participants in a safe, reliable and irreversible way. One of its strengths is that it does not require the intervention of third parties to establish relationships of trust between the parties. This is possible thanks to the fact that the information is distributed in different nodes that are independent of each other, since they register and validate it without the need for trust between them. One of the great advantages of the Blockchain technology is that the transaction data, once registered, is impossible to falsify.*

*One of the sectors in which this technology has the most applicability is transportation. This Final Master's Project is focused on making a decentralized web application on the Ethereum platform, which generates a route or tour, including different means of transport to move from point A to point B.*

**Keywords:** Blockchain, smart contract, Ethereum, security.

[2] computer programs that are programmed to automatically execute agreements that have been determined by two or more parties.

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. Motivación

La aparición de la tecnología Blockchain y su popularidad obtenida sobre todo por las criptomonedas (bitcoin, Ether, etc.), ha provocado que varios sectores de la sociedad muestren interés por esta tecnología. Desde su origen a partir de la base establecida en el año 1991 por Stuart Haber y W. Scott Stornetta [1] hasta la actualidad se han realizado numerosos avances. Entre ellos destaca la aparición de los contratos inteligentes, que ha permitido crear proyectos basados en Blockchain en áreas como la sanidad o la logística, mostrando así el amplio abanico de usos que se le puede dar a esta tecnología. A continuación se mencionan algunas de las soluciones de Blockchain en esas áreas:

- **Sector sanitario**
  - **HealthCombix:** Para proveer servicios médicos descentralizados [14] (ver Figura 1.1).
  - **Patientory:** Para registros personales de salud [22] (ver Figura 1.2).
  - **iSolve:** Para desarrollo de fármacos y suministros de medicamentos [18] (ver Figura 1.3).

Figura 1.1: Logo de HealthCombix

Figura 1.2: Logo de Patientory


Figura 1.3: Logo de iSolve

- **Sector Industrial:** Gigantes como Airbus o Daimler [25] han comenzado a trabajar en la trazabilidad de las piezas a lo largo de todo su ciclo de vida. Estas empresas sabían por donde pasaba cada pieza dentro de su planta, pero una vez estas salen de su almacén, debían fiarse de lo que les dijera el proveedor. Ahora las empresas industriales pueden hacer que sus proveedores registren en Blockchain el código de cada pieza y añadan todos los datos asociados, incluyendo las correcciones que hagan falta.
- **FoodTrust:** Para gestión de la cadena de suministros por IBM y Walmart [16] (ver Figura 1.4).

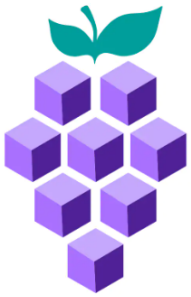

Figura 1.4: Logo de FoodTrusted

- **Hyperledger Indy:** Proporciona herramientas, bibliotecas y componentes reutilizables para proporcionar identidades digitales enraizadas en cadenas de bloques, de modo que sean interoperables a través de dominios administrativos, aplicaciones, etc. Indy es interoperable con otras cadenas de bloques o puede utilizarse de forma autónoma impulsando la descentralización de la identidad [15] (ver Figura 1.5).

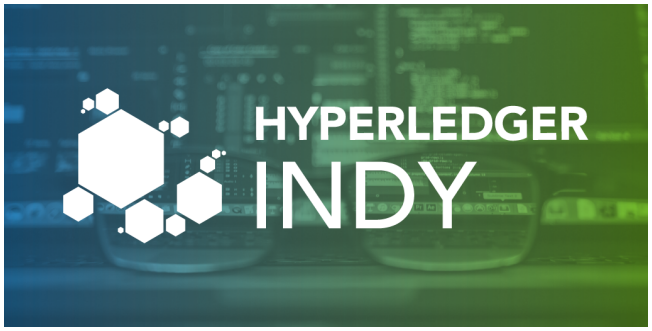


Figura 1.5: Logo de Hyperledger Indy

- **Agora:** Es un ecosistema de votación basado en cadenas de bloques que permite a cualquier persona en cualquier lugar votar en línea desde un dispositivo digital de forma totalmente segura y fácil [24] (ver Figura 1.6).


Figura 1.6: Logo de Agora

Los contratos inteligentes o Smart Contracts [12] son la aplicación más utilizada de la tecnología Blockchain en los tiempos actuales. Este concepto fue introducido en el año 1994 por Nick Szabo, un estudioso legal y criptógrafo. Un sistema basado en Blockchain posibilita la interacción entre todas las entidades en la red entre sí de forma distribuida, eliminando así el requisito de cualquier tercero de confianza. Los contratos van almacenados en la Blockchain y se ejecutan de manera automática y autónoma, de acuerdo a una serie de parámetros previamente programados (todo de forma inmutable, transparente y completamente segura). El Smart Contract, al eliminar cualquier tipo de intermediario, simplifica los procesos y, además, ahorra costes al consumidor.

Tras haber comentado un poco cómo es la tecnología Blockchain y lo que se puede hacer con ella mediante los diferentes aplicativos que hay en la actualidad, resulta interesante poder aplicarla en el sector del transporte, por lo que este TFM está enfocado en ello. Se ha realizado una aplicación web descentralizada en la que un usuario indica un punto origen y un punto destino, y la aplicación crea una posible ruta con diferentes transportes, y el coste total que tiene que pagar para realizar dicha ruta, usando la criptomoneda de Ethereum, que es el Ether.

## 1.2. Objetivos y competencias del proyecto

Como ya se ha comentado, el objetivo de este proyecto es el diseño e implementación de una aplicación web descentralizada para el sector de transportes, usando la tecnología Blockchain y la plataforma de Ethereum. Teniendo en cuenta esta idea, se describen a continuación algunas de las tareas de este trabajo:

- **Creación de la ruta:** El usuario elige dos puntos (origen y destino) y se crea la ruta que debe tomar, junto al transporte correspondiente, mostrándole el coste total de la misma. Este coste se calcula sumando el coste de los diferentes transportes a tomar (para la selección de los transportes, se escogen los que tengan menor coste).
- **Designación de transportes:** Se eligen qué transportes se usan en la aplicación y las preferencias a la hora de crear la ruta.
- **Costes de los transportes:** Se designa qué coste tienen los transportes cada vez que se genere una ruta.
- **Actualización de la ruta:** Se actualiza la ruta a medida que se vayan tomando los transportes, es decir, se marca la ruta que ya se ha recorrido y la que falta por recorrer.

- **Registro de usuario/transporte:** Cuando se esté con una cuenta que no esté registrada en la aplicación, se deberá registrar según lo que la persona que esté activa en el momento decida.
- **Visualización de los transportes:** Se visualizarán los transportes que estén registrados en la aplicación.

## 1.3. Planificación del proyecto

En esta parte se describe la planificación que se ha llevado a cabo para la creación del proyecto. Se han definido diferentes fases para cubrir los diferentes aspectos, desde su planteamiento hasta su finalización. Estas fases son las siguientes:

- **Análisis de tecnologías y antecedentes:** Se ha realizado un análisis de las diferentes tecnologías nombradas con anterioridad(Blockchain, Smart Contract, etc.), además de investigar antecedentes actuales que usen dichas tecnologías que estén implantadas con éxito en los diferentes sectores.
- **Formación y comprensión de las tecnologías y el lenguaje de programación Solidity:** Se ha buscado información acerca de la tecnología y el lenguaje de programación Solidity para ver cómo es su funcionamiento, usando para ellos Smart Contracts sencillos, además de realizar el ejemplo **Pet Shop** de Truffle Suite (Ethereum) usando dichas tecnologías.
- **Selección de las herramientas:** Se ha seleccionado las herramientas a usar para el desarrollo de la aplicación teniendo en cuenta el objetivo de la misma y las tecnologías previas que se han nombrado.
- **Implementación de la aplicación:** Se ha llevado a cabo la implementación de la aplicación web teniendo en cuenta todo lo anterior.
- **Desarrollo de la memoria:** Finalmente se ha pasado a desarrollar la memoria de todo el trabajo realizado y el resultado final que se ha obtenido.

En la figuraA.3 se puede ver un esquema de cómo es el esquema de un ruta que un usuario toma en la aplicación.

## 1.4. Estructura de la memoria

La memoria se ha estructurado de la siguiente manera:

- **Capítulo 2.** Se aportan detalles de la tecnología Blockchain, los Smart Contracts y la plataforma Ethereum, que es donde se ha desarrollado la aplicación.
- **Capítulo 3.** Se nombran las tecnologías y herramientas usadas para el desarrollo de la aplicación.
- **Capítulo 4.** En este capítulo se habla de la aplicación desarrollada, las interfaces que tiene y los contratos inteligentes que se han elaborado.
- **Capítulo 5 y 6.** En estos capítulos se encuentran las conclusiones de todo el trabajo realizado en este proyecto y la visión de futuro que se tiene del mismo.
- **Capítulo 7.** En este capítulo se aporta un análisis sobre el coste total del proyecto.

# Capítulo 2

# Antecedentes

A lo largo de este capítulo se aportan detalles de la tecnología Blockchain, además de todo lo que la rodea (la criptografía que hay, funciones hash, etc.), y se extenderá un poco más en el concepto de contrato inteligente y su importancia en el uso de cadenas de bloques. También se nombrará la plataforma Ethereum que es donde se ha desarrollado la aplicación.

## 2.1. Criptografía

Lo primero que se explica es la criptografía en la que se basa la tecnología Blockchain, para entender cómo funciona y la seguridad que tiene, pues la mayor parte basada en inmutabilidad e integridad de los datos que se guardan en ella.

### 2.1.1. Función hash

Se llama hash [8] al resultado de una función hash, la cual es una operación criptográfica que genera identificadores únicos e irrepetibles a partir de una información dada. El hash es una pieza clave en la tecnología Blockchain, además de tener una amplia utilidad.

La aparición de la primera función hash fue en el año 1961. Wesley Peterson creó la función CRC (Cyclic Redundancy Check), creada para comprobar cómo de correctos eran los datos transmitidos en redes (como Internet) y en sistema de almacenamiento digital. Fácil de implementar y muy rápida, ganó bastante aceptación. Con la evolución de la informática y los ordenadores, estos sistemas se fueron especializando cada vez más (surgió el MD5, SHA, SHA-256, SHA-3, etc.). Además de esto, las funciones hash tienen las siguientes características:

- **Fáciles de calcular:** Los algoritmos de hash son muy eficientes y no requieren de grandes potencias de cálculo para ejecutarse.
- **Generan compresión:** Esto quiere decir que, sin importar del tamaño de la entrada de datos, el resultado siempre será una cadena de longitud fija. En el caso de la función SHA-256, la cadena tendrá una longitud de 64 caracteres.
- Cualquier mínimo cambio en la entrada de datos genera un hash distinto al hash generado con la entrada de datos original.
- **Son irreversibles:** Tomar un hash y obtener los datos que generaron ese hash no es posible en la práctica. Esto es uno de los principios que hacen a las funciones hash seguras.

Gracias a que las funciones hash tienen estas características, se emplean dentro de la Blockchain para que, de esta manera, se asegure la integridad de los datos. Como el hash

de cada bloque va almacenado en el siguiente, si un bloque se modifica, su hash cambiará por lo que se detectaría que está pasando algo raro.

Cabe destacar que a pesar de que las funciones hash actuales tienen un alto nivel de seguridad, no significa que sean infalibles. Por ejemplo, la función MD5 se pudo romper en el año 1996. Con ello quedó obsoleta y se recomendó abandonar su uso. Sin embargo, para la función SHA-256, por ejemplo, se calcula que para romper su seguridad harían falta miles de años usando supercomputadores actuales. Esto se debe a que la longitud del hash MD5 es de 128 bits, mientras que la longitud del SHA-256 es el doble.

En la Figura 2.1 se puede observar cómo es el esquema general de una función hash:

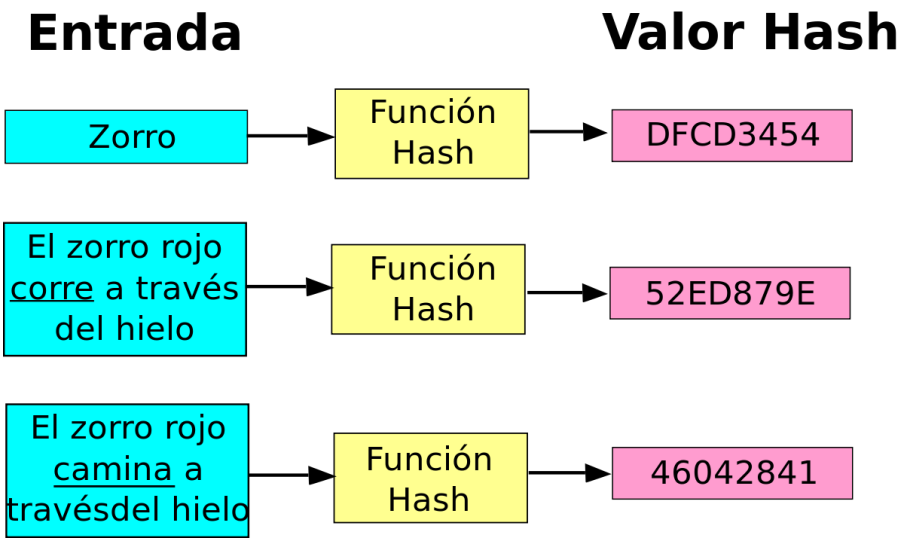


Figura 2.1: Función hash

### 2.1.2. Criptografía asimétrica o de clave pública

La criptografía asimétrica o de clave pública [4] consiste en el uso de una fórmula matemática muy compleja para crear un par de claves. Esta primera clave es la **clave privada**, que es de uso exclusivo para el creador del par de claves, y sirve para descifrar el mensaje de forma completamente segura. La segunda clave es la **clave pública**, que es la que se crea a partir de la clave privada, de forma que el creador de esa clave puede compartir la clave pública resultante y gracias a ella quienes la tengan podrán enviarle información cifrada, que sólo será accesible usando la correspondiente clave privada, pareja de esa clave pública.

La criptografía asimétrica nació de la mano de Ralph Merkle, Whitfield Diffie y Martin Hellman en el año 1976. Se trata de un esquema basado en información privada e información pública que garantiza el establecimiento de comunicaciones completamente seguras, incluso sobre canales inseguros. En la Figura 2.2 se puede ver cómo es la estructura de la generación de claves entre dos sujetos o entidades A y B:

Una vez se realizan las operaciones mostradas en la Figura 2.2, tanto A como B tienen cada uno sus propias clave privada y pública, y una clave secreta $K$ que comparten ambos.

El concepto tras el esquema de Diffie-Hellman conlleva el nacimiento de la criptografía asimétrica en base a que cada usuario posee un par de claves: su clave privada (que solo él conoce) y su clave pública (que es calculada a partir de la clave privada y compartida en un canal inseguro con otros usuarios). Este tipo de criptografía permite que se puedan realizar las siguientes acciones:

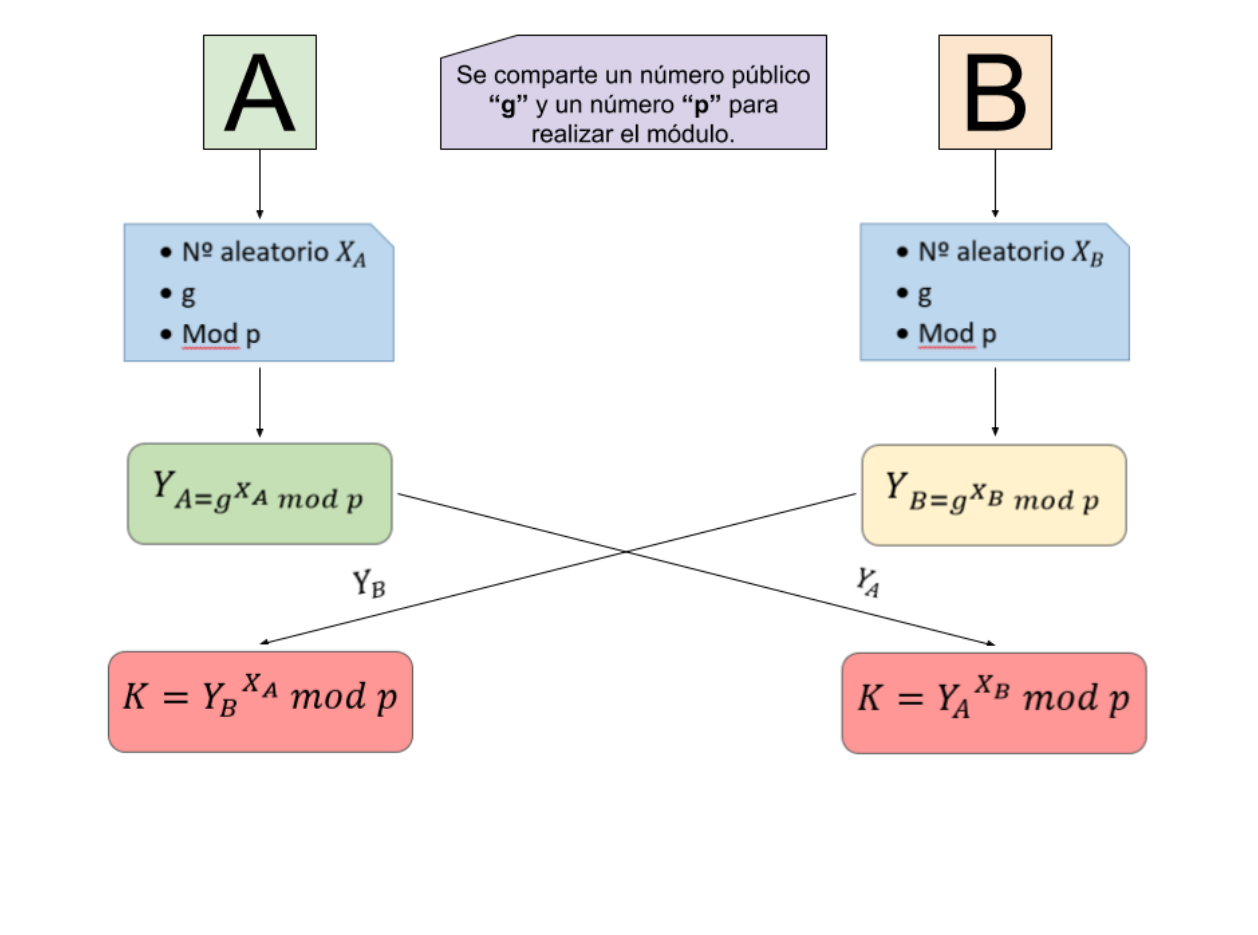


Figura 2.2: Protocolo criptográfico de Diffie-Hellman

- **Envío de mensajes cifrados:** La usuaria A puede enviarle mensajes cifrados al usuario B. Para ello, lo que tiene que hacer es cifrar el mensaje usando la clave pública del receptor. Una vez se tiene el mensaje cifrado, la usuaria A lo puede enviar por un canal inseguro ya que sólo el usuario B puede descifrar el mensaje usando su propia clave privada. En la Figura 2.3 se puede observar cómo sería un esquema de cifrado de clave pública.

- **Firma digital de documentos:** La usuaria A quiere enviarle un documento al usuario B indicándole que es él quien le mandó ese documento (es decir, que es quien dice ser). Para ello, la usuaria A coge su propia clave privada y cifra con ella el mensaje. Ese mensaje cifrado lo envía al usuario B, de forma que éste puede verificar que la usuaria A es quien envió el mensaje (y sólo ella pudo ser ya que es la única que tiene su clave privada). Además de esto también se garantiza que el mensaje no ha sido alterado por un tercero, ya que cualquier modificación sería detectable. En la Figura 2.4 se puede observar cómo funciona una esquema de firma digital.

Estas dos funcionalidades que da la criptografía asimétrica proporciona **confidencialidad, autenticidad e integridad** en los datos. En Blockchain se usa una clave pública(direcciones) que es compartida con el resto de los nodos, y una clave privada(un número escogido al azar) a la que sólo tiene acceso el usuario. Con la clave privada se firman las transacciones que el usuario realice, para verificar posteriormente que es ese usuario quien las hizo(cada vez que una transacción se añade a un bloque, se firma con la clave privada del usuario). Los demás usuarios pueden verificar que esas transacciones han sido realizadas por el primer

usuario usando la clave pública(que es la dirección de la transacción. Si se cambia cualquier dato de la transacción, la firma no sería igual y se detectaría inmediatamente que pasa algo raro).


Figura 2.3: Cifrado asimétrico

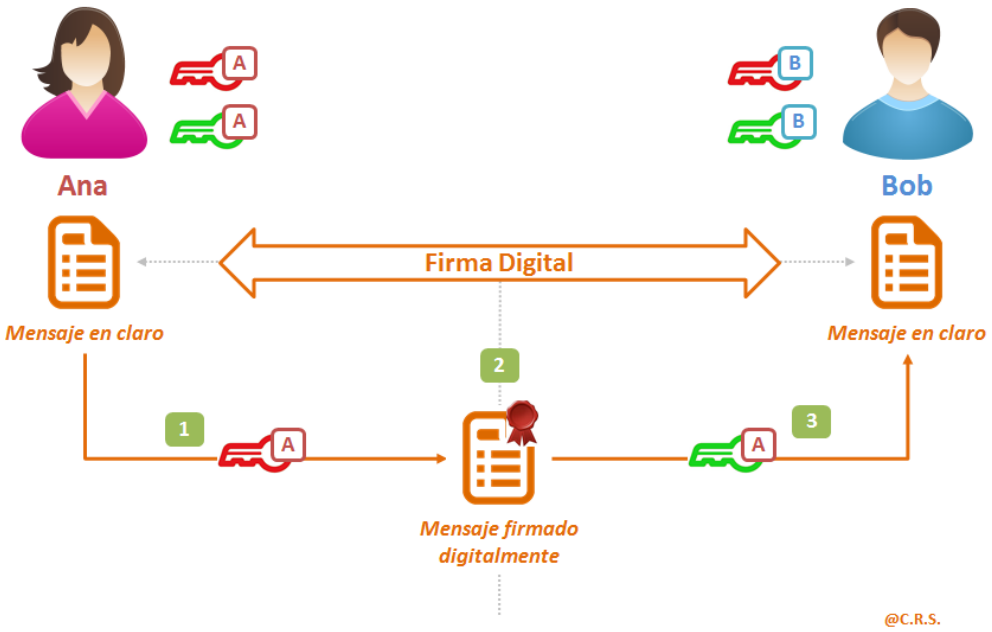


Figura 2.4: Firma digital

### 2.1.3. Árboles de Merkle

Un árbol de Merkle [10] es una estructura de datos estratificada que tiene como finalidad relacionar cada nodo con una raíz única asociada a éste. Para ello cada nodo tiene que estar identificado con un hash. Estos nodos iniciales, llamados nodos hojas, se asocian con un nodo superior llamado nodo padre. Este nodo padre tendrá a su vez un identificador único como resultado del hash de sus nodos hijos. Esta estructura se repite hasta llegar al nodo

raíz o raíz Merkle, cuyo identificador está asociado a todos los nodos del árbol.

Gracias a esta estructura única, los árboles de Merkle permiten relacionar una gran cantidad de datos en único punto, que es la raíz Merkle. De esta forma, la verificación y validación de esos datos puede pasar a ser muy eficiente, ya que sólo tiene que verificar la raíz del árbol Merkle, en lugar de mirar toda la estructura. Este diseño, que fue creado por Ralph Merkle en el año 1979 con el fin de agilizar el proceso de verificación de grandes cantidades de datos, tiene las siguientes propiedades:

- Son bastantes eficientes a la hora de verificar grandes cantidades de información, ya que, como se dijo, sólo se mira la raíz del árbol y no toda la estructura.
- Ofrece una gran adaptabilidad a distintos problemas informáticos (software de base de datos, sistemas de archivos, estructuras de claves públicas, redes distribuidas P2P, etc.).
- Son bastante eficientes en la sincronización de datos, ya que no se necesita comparar todos los datos para descubrir que algo ha cambiado, sino que hay que comparar solo el hash de los árboles. Una vez se identifique qué nodos hijos han cambiado, esa porción de datos se envía a través de la red para sincronizarse en todos los nodos.

Tras ver estas características de cómo es un árbol de Merkle, no cabe duda de por qué son tan usados en Blockchain, más aún cuando se trabaja con una gran cantidad de datos y se necesite acceder a ellos de la manera más eficiente posible, ya que no es necesario recorrer toda la estructura, sino mirar la raíz del árbol. En las Figuras 2.5 y 2.6 se puede ver ilustrado todo lo comentado:

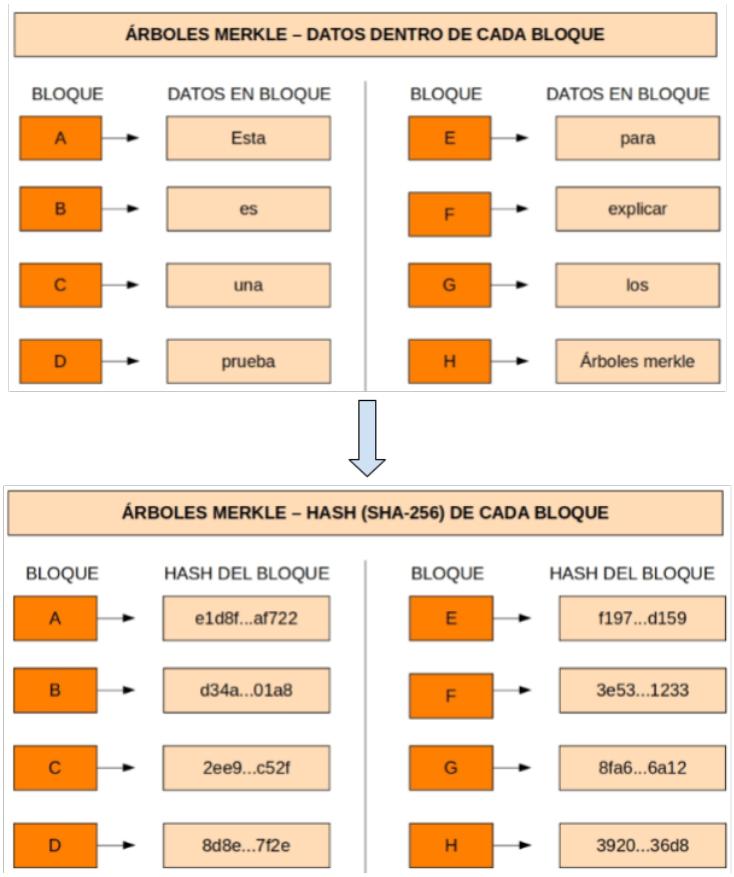


Figura 2.5: Hash de los datos de los nodos del árbol

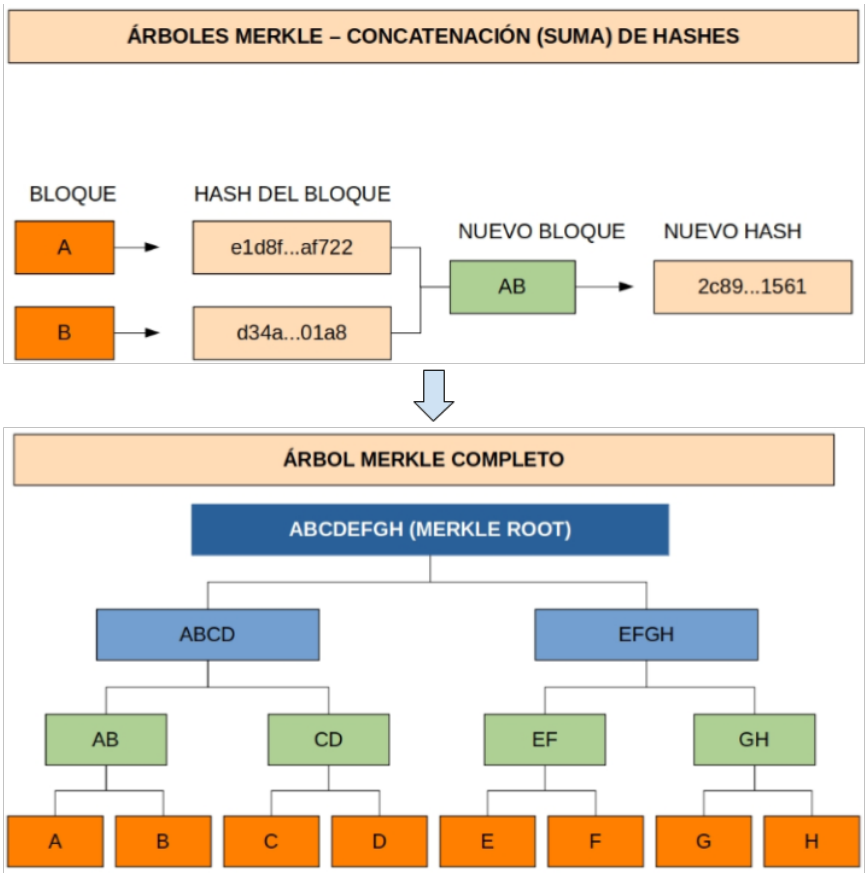


Figura 2.6: Concatenación de los nodos y creación del árbol Merkle

## 2.2. Blockchain

La tecnología Blockchain [21] generalmente se asocia con el bitcoin y otras criptomonedas, pero eso solo es la punta del iceberg. Esta tecnología, cuyo origen es en 1991, cuando Stuart Haber y W. Scott Stornetta describieron el primer trabajo sobre una cadena de bloques asegurados criptográficamente, no fue notoria hasta 2008, cuando se hizo popular con la llegada del bitcoin.

La cadena de bloques (Blockchain) es un registro único, consensuado y distribuido en varios nodos de una red. Su funcionamiento puede parecer complejo pero la idea básica es sencilla. En cada bloque se almacena lo siguiente:

- Los registros o transacciones válidas.
- Información que haga referencia al bloque.
- El enlace que tiene dicho bloque con el bloque anterior y con el siguiente. Este enlace se hace a través del hash que tiene cada bloque, un código único que funciona como la huella digital del bloque.

Cada bloque tiene un lugar específico e inamovible dentro de la cadena, ya que cada uno contiene el hash del bloque anterior. La cadena completa se guarda en cada nodo de la red que conforma la Blockchain, por lo que se almacena una copia exacta de la cadena en todos los participantes de la red de nodos.

En la Figura 2.7 se puede observar cómo es el esquema de la Blockchain, donde se tienen diferentes bloques en los que cada uno cuenta con la información mencionada.

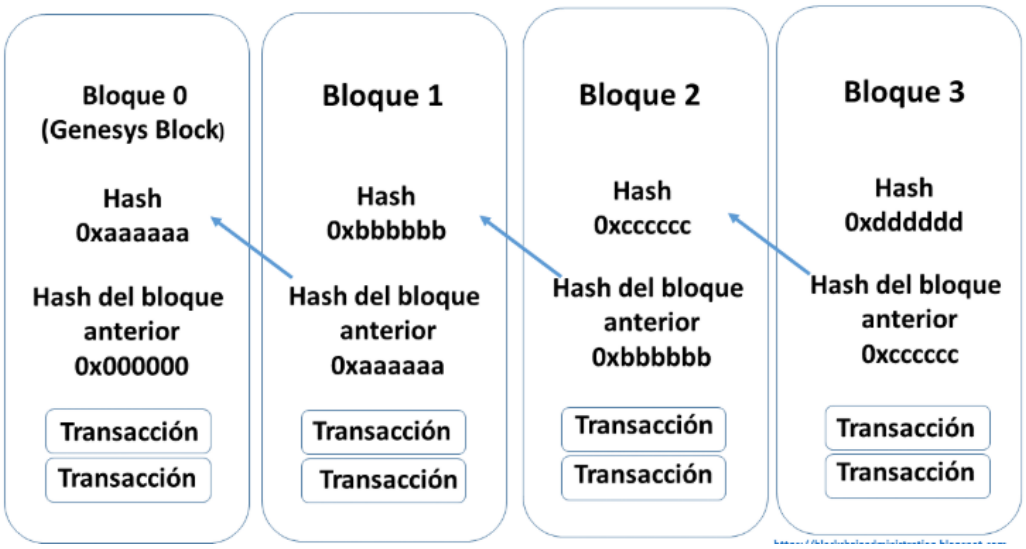


Figura 2.7: Esquema de la Blockchain

### 2.2.1. Bloques

En la Blockchain [20], aparte de su huella digital (hash), cada bloque contiene información adicional que es relevante. En esta sección en particular se analiza el contenido de un bloque de Ethereum, que es en la plataforma donde se ha desarrollado la aplicación. En cada plataforma los bloques contienen una información u otra, aunque a rasgos generales hay campos que se repiten. A continuación se nombra cada uno de estos campos habituales:

- **Height:** Hace referencia a la altura en la que se encuentra el bloque, es decir, el número de este.
- **Timestamp:** Es la marca temporal, cuando se creó el bloque con todo detalle. Por ejemplo: ."Este bloque se creó el 20 de Noviembre de 2019 a las 11:30 y 49 segundos AM (de la mañana) de la zona horaria +UTC, que significa +0 horas de la Hora Universal Coordinada (UTC, del inglés 'Universal Time Coordinated')".
- **Transacciones:** Son las operaciones realizadas en la cadena de bloques. Puede ser desde el envío de tokens de un usuario a otro, a operaciones del contrato inteligente. Cada una de estas transacciones está firmada por el usuario que la realiza.
- **Hash:** Es un proceso por el cual unos datos (texto, imágenes, lista de transacciones, etc.) se procesan en otra pequeña pieza de datos que es totalmente diferente y aleatoria. Los hashes se usan normalmente para crear un identificador único para un documento en particular que no puede ser falsificado. Generalmente, cambiar incluso una letra en un documento, genera un hash totalmente diferente, por ejemplo el hash de la palabra "vaca"puede ser "d38bbc8e93c09c09f6ed3fe39b5135da91ad1a99d397d397ef16948606cdcbd14929f9d", mientras que el hash de "baca"puede ser "b4013c0eed56d5a0b448b02ec1d10dd18c1b3832068fbbdc65b98fa9b14b6dbf".
- **Hash padre:** Este dato se refiere al hash del bloque anterior y sirve para seguir ese orden dentro de la cadena de bloques.
- **Nonce/Proof of Work Nonce** [13]: Es un campo cuyo valor se establece de forma que el hash del bloque contenga una cadena de ceros. Cualquier cambio en el bloque de datos hará que el bloque de hash sea completamente diferente.

Una vez analizado cómo está compuesto un bloque, se tienen los ingredientes de la seguridad que aporta la Blockchain, ya que como un bloque almacena la información del hash del bloque anterior, si se modificara uno se detectaría inmediatamente porque su hash no coincidiría con el hash que tiene guardado el siguiente bloque. Para que esto no se detectara, el atacante tendría que modificar todos los bloques posteriores al que le interesa modificar, lo que implicaría un coste computacional inabarcable.

Este coste computacional, que está asociado al algoritmo de consenso Proof of Work (PoW) que se explicará más adelante, se debe a que el atacante tiene que calcular el valor del nonce (que es lo que se conoce como minado). En otras palabras, para que incluya un bloque primero tiene que minarlo para hallar el valor nonce que hace que el hash de dicho bloque comience con el número determinado de ceros (que es establecido por la dificultad de minado del bloque). A continuación se muestra cómo se vería un bloque con todo lo comentado anteriormente en la Figura 2.8:

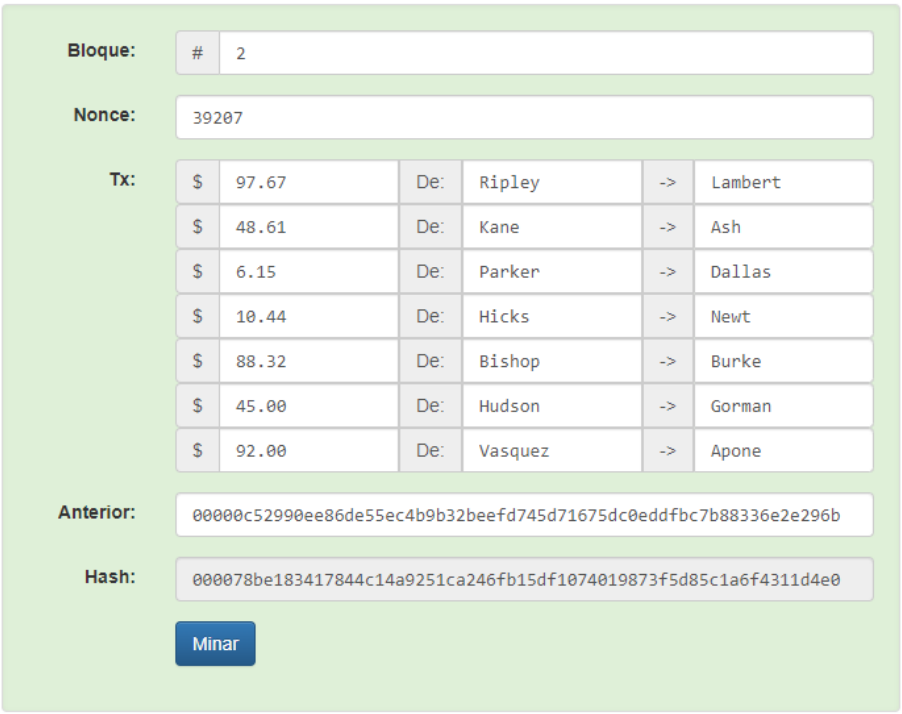


Figura 2.8: Bloque de ejemplo

### 2.2.2. Transacciones

En esta sección se ve cómo se realizan las transacciones en la cadena de bloques paso por paso (usando las claves que se crean en en los procesos de la criptografía asimétrica explicada), además de ilustrar en la Figura 2.9 esos mismos pasos para apoyar la explicación:

1. Alice desea realizar una transacción con Bob. Para ello, cada uno de los datos necesarios en la transacción se pasan a una función de firma (junto con la clave privada de Alice y la clave pública de Bob). Esto permite devolver la firma de la transacción.

2. Esa firma que se ha generado, junto con la clave pública de Alice y los datos de la transacción se envían a la red donde están todos los nodos. Estos se encargarán de validar la transacción (es decir, comprobar que la emisora es quien dice ser y que se cumple con todo lo necesario para realizar la transacción). Para realizar esta verificación

se usa lo que se envió a la red (es decir, clave pública, firma y datos). Comprobando la firma se puede ver si la transacción es válida o no.

3. Si la transacción es válida, los nodos encargados de crear bloques (llamados mineros) la añaden al nuevo bloque que se está creando, para que cuando se termine de construir, se envíe a la red de nodos.

4. El bloque, una vez se envía a la red, es aceptado por los nodos sólo si todas las transacciones que contiene son válidas y ejecutables.

5. Una vez se comprueba la validez del bloque, los nodos que lo han comprobado lo añaden a su copia de la cadena de bloques que tienen en local, y continúan con los siguientes bloques, añadiendo como hash previo del nuevo bloque que se vaya a añadir, el hash del bloque que ya se añadió, ejecutando de esta manera la transacción al completo.

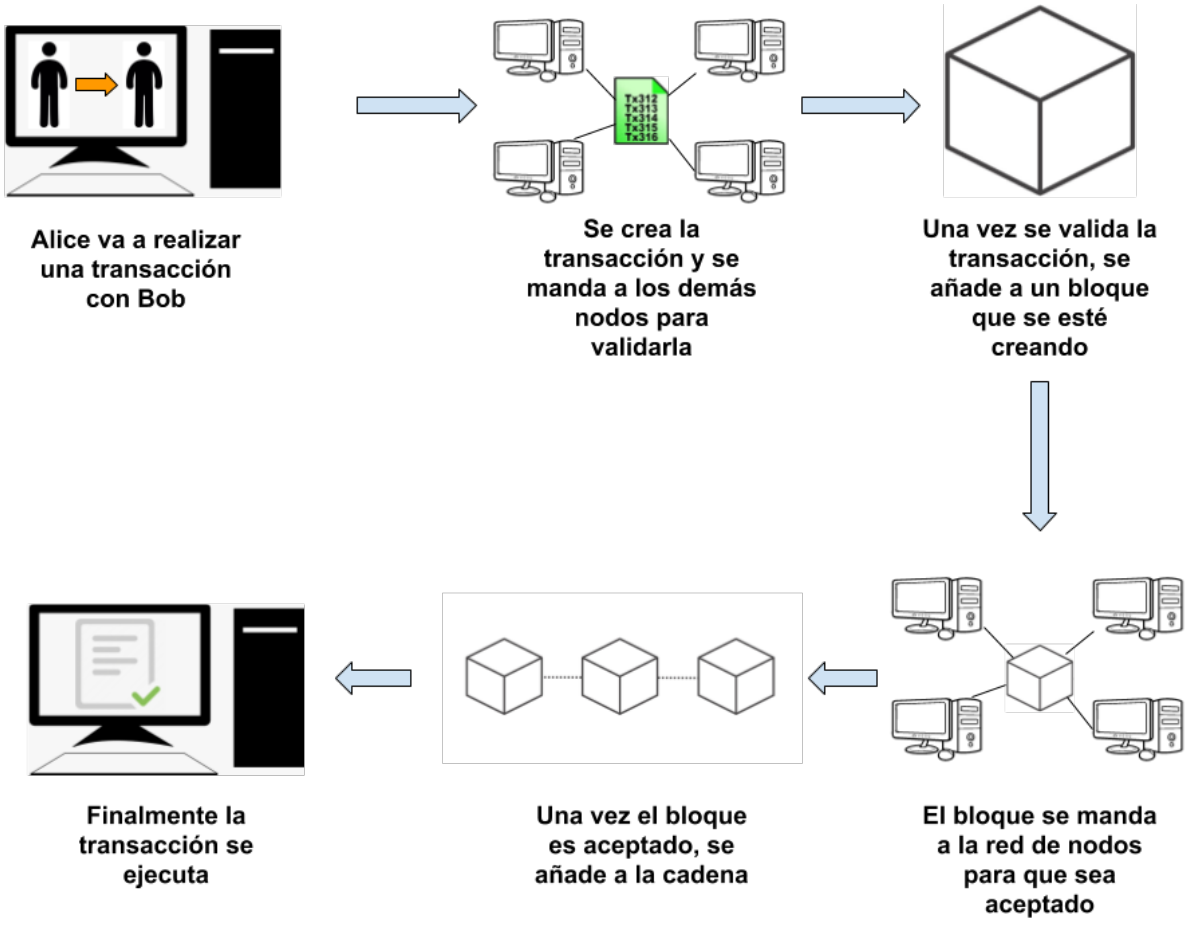


Figura 2.9: Esquema de cómo se realiza una transacción

### 2.2.3. Algoritmos de consenso

Un algoritmo de consenso [2] puede ser definido como el mecanismo que se realiza en la Blockchain para alcanzar consenso. Las Blockchain descentralizadas se erigen como sistemas distribuidos, y como no se tiene una unidad central, los nodos de la misma también están distribuidos, por lo que se necesita alguna manera de que se pongan de acuerdo respecto a la validez de las transacciones que se realizan. Es aquí donde entra en juego los algoritmos de

consenso, ya que se encargan de que las reglas del protocolo [1] son respetadas y garantizan que todas las transacciones tienen lugar de una forma fiable.

De los algoritmos de consenso que hay disponibles, el más usado es el que se conoce como prueba de trabajo (Proof of Work o PoW). El algoritmo de PoW [7] es el más conocido y antiguo algoritmo de consenso y consiste en que las partes de una red deben realizar con éxito un trabajo computacionalmente costoso para poder acceder a los recursos de dicha red. Este algoritmo funciona bajo el concepto de requerir un trabajo al cliente (normalmente consiste en realizar complejas operaciones de cómputo), que luego es verificado por la red.

Este coste computacional está enfocado a la creación de nuevos bloques en la Blockchain. Para ello lo que hacen los usuarios (también llamados mineros) es buscar el valor del campo **nonce** de la cabecera, para que luego al aplicar una función hash sobre esta cabecera se obtenga un hash que empiece por un número determinado de ceros (como se explica en la subsección 2.2.1). Este esfuerzo que ha realizado el nodo que ha encontrado el valor de nonce válido se conoce como **prueba de trabajo**, y es recompensado para animarle a seguir apoyando a la red. Una vez se tiene el bloque creado, no puede ser alterado a menos que se

vuelva a encontrar un valor adecuado para el campo nonce. Cuantos más bloques se vayan encadenando en la Blockchain, mayor esfuerzo se requerirá para cambiar un bloque, por lo que también se necesitará mayor coste computacional para poder minar dicho bloque y los siguientes. Este es uno de los problemas que tiene el algoritmo prueba de trabajo.

Además del algoritmo PoW que se ha nombrado, existen otros dos que son bastante usados en la actualidad:

- **Prueba de participación (Proof of Stake)** [11]: Este algoritmo, al igual que la prueba de trabajo, busca llegar al consenso pero usando operaciones computacionales más sencillas. A diferencia de la prueba de trabajo, el algoritmo prueba de participación elige al creador de un nuevo bloque de manera determinista, dependiendo de la riqueza que tenga acumulada en la red, también conocida como stake.
- **Prueba de autoridad (Proof of Authority)** [6]: Este algoritmo está enfocado para usarlo en redes privadas. El rendimiento que tiene frente a los dos algoritmos vistos anteriormente es más eficiente, ya que se tienen en cuenta las identidades reales de los propietarios de los nodos. En el algoritmo se elige un conjunto de estas identidades que sean de confianza para que se encarguen de ser los únicos en añadir nuevos bloques a la cadena. Este algoritmo, además de ser el que más eficiencia tiene, también proporciona más escalabilidad a la red. Es usado en redes privadas donde se requiera velocidad en las operaciones.

### 2.2.4. Tipos de Blockchain

Actualmente se pueden diferenciar tres tipos de Blockchain [3]:

- **Blockchain pública:** Este fue el primer tipo de Blockchain que existió, y se refiere a las Blockchains que se encuentran públicamente accesible desde Internet. Un ejemplo de este tipo de Blockchain son **bitcoin, Ethereum**, Dash, Monero o Zcash. Este tipo de Blockchain mantienen abierto al público sus datos, el software y su desarrollo, de forma que cualquier persona puede revisar, auditar, desarrollar o mejorar los mismos. Cualquier persona puede formar parte de la misma (ya sea como usuario, minero o

[1] Reglas primarias de una Blockchain, como puede bitcoin o Ethereum.

administrador de un nodo). El funcionamiento que hay en este tipo de cadenas es completamente transparente y abierto, cualquier persona puede revisar o auditar el funcionamiento de la red y su software. Este tipo de redes son descentralizadas y no existe ningún tipo de autoridad central que regule su funcionamiento. El mantenimiento económico de la misma depende del sistema integrado que tenga (lo mas normal es que el sistema dependa de la minería y el cobro de comisiones por cada transacción que se realice dentro de la red).

- **Blockchain privada o permisionada:** Más tarde, con la evolución de la tecnología Blockchain y su expansión, muchas empresas se interesaron en ella. Esto derivó en el desarrollo de soluciones Blockchain privadas o permisionadas. Este tipo de Blockchain generalmente cuenta con los mismos elementos que una Blockchain pública, pero a diferencia de éstas, las Blockchain permisionadas dependen de una unidad central que controla todas las acciones dentro de la misma. El acceso ahora está restringido a elementos que solo pueden ser autorizados por la unidad de control central, no está abierto a todo el mundo. El acceso a la información generado por la Blockchain es privado, no cualquiera puede verlo. En cuanto al mantenimiento de este tipo de Blockchain, generalmente la empresa que sostenga el proyecto es la que se encarga. Normalmente, este tipo de redes privadas no cuentan con criptomonedas ni se realizan acciones de minería. Ejemplos de Blockchain privadas que se pueden encontrar son **Hyperledger** (que es una de las redes privadas más importantes que hay, iniciada por la fundación Linux), **Corda** (del consorcio R3) y **Quorum** (de JPMorgan).

- **Blockchain híbrida o federada:** Este tipo de Blockchain es una fusión entre las dos que se nombraron anteriormente. Es un intento de aprovechar lo mejor de ambos mundos. En estas Blockchain, la participación en la red es privada. Es decir, el acceso a los recursos de la red es controlado por una o varias entidades. Sin embargo, el libro de contabilidad o cualquier tipo de información generada por al Blockchain es accesible de forma pública. Esto significa que cualquier persona puede explorar bloque a bloque todo lo que sucede en dicha Blockchain. En este tipo de redes no existe la minería ni las criptomonedas, el consenso se da por otros medios que aseguran que los datos son correctos. Este tipo de Blockchain es parcialmente descentralizado, lo que conlleva un mejor nivel de seguridad y transparencia. Algunos ejemplos de Blockchain híbrida son **BigchainDB** (un proveedor de tecnología Blockchain) o **Evernym**, una Blockchain híbrida que quiere facilitar la gestión de la Identidad Digital Soberana (o Self Sovereign Identity).

### 2.2.5. Ethereum

Por último, para dar por finalizado este capítulo se describe brevemente lo que es la plataforma Ethereum2.10, que es donde se ha desarrollado este proyecto.

Ethereum [17] es una plataforma digital que adopta la tecnología de cadena de bloques (Blockchain) establecida por bitcoin y expande su uso a una gran variedad de aplicaciones. No debe confundirse con Ether (la criptomoneda subyacente de la red) a la que habitualmente también se la denomina Ethereum.

La plataforma Ethereum fue creada en 2015 por el programador Vitalik Buterin, con la perspectiva de crear un instrumento para aplicaciones descentralizadas y colaborativas. Ether (ETH) es un token que puede ser utilizado en transacciones que usen este software.

La Blockchain de Ethereum es muy similar a la de bitcoin, pero su lenguaje de programación

permite a los desarrolladores crear software a través del cual gestionar las transacciones y automatizar ciertos resultados. Este software se conoce como **contrato inteligente** o Smart Contract.

Si un contrato tradicional describe los términos de una relación, un contrato inteligente se asegura de que esos términos se cumplan escribiéndolos en código. Un Smart Contract es un programa que automáticamente ejecuta el contrato una vez que las condiciones predefinidas se cumplen, eliminando el retraso y el coste que existe al ejecutar un acuerdo de manera manual.

Por poner un ejemplo sencillo, un usuario de Ethereum podría crear un contrato inteligente para enviar una cantidad establecida de Ether a un amigo en una fecha determinada. Escribirían este código en la cadena de bloques y cuando el contrato se complete (es decir, cuando se alcance la fecha acordada) los Ether se enviarán automáticamente.

Figura 2.10: Logo Ethereum

# Capítulo 3

# Tecnologías

En este capítulo se habla sobre las tecnologías usadas para desarrollar la aplicación propuesta.

## 3.1. JavaScript

Para realizar la aplicación web se usó el lenguaje **JavaScript**, (ver Figura 3.1). Es un lenguaje de programación interpretado [26]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Este lenguaje es el usado por excelencia para el desarrollo web, y está integrado con muchos framework[1] para conseguir este objetivo (React, Vue, etc.). En la siguiente sección se hablará del framework de desarrollo web que se eligió para este proyecto.


Figura 3.1: Logo de JavaScript

## 3.2. React

Como se dijo anteriormente, ahora se pasará a hablar sobre el framework elegido para el desarrollo de interfaces gráficas de usuario. Este framwork es **React** (ver Figura 3.2).

React [9] es una librería desarrollada inicialmente por Facebook. Es de software libre y a partir de su liberación está acaparando una creciente y ya amplia comunidad de desarrolladores. Además de facilitar el desarrollo ágil de componentes de interfaces de usuario, el requisito principal con el que nació React era ofrecer un elevado rendimiento, mayor que otras alternativas existentes en el mercado.

Sirve para desarrollar aplicaciones web de una manera más ordenada y con menos código

[1] Un framework para aplicaciones web es un framework diseñado para apoyar el desarrollo de sitios web dinámicos, aplicaciones web y servicios web. Este tipo de frameworks intenta aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web.

que si usas JavaScript puro o librerías como jQuery centradas en la manipulación del DOM. Permite que las vistas se asocien con los datos, de modo que si cambian los datos, también cambian las vistas. La interfaz de usuario se crea a partir de un componente, el cual encapsula el funcionamiento y la presentación. Además de esto hay componentes que se basan en otros, es decir, los usan dentro de ellos para solucionar problemas de la aplicación que sean más complejos. También permite que el desarrollador cree su propio componente(usando varios componentes que ya tenga React si así lo quiere y aplicando sus propios estilos) para que de esta manera se tenga el código más limpio y ordenado.

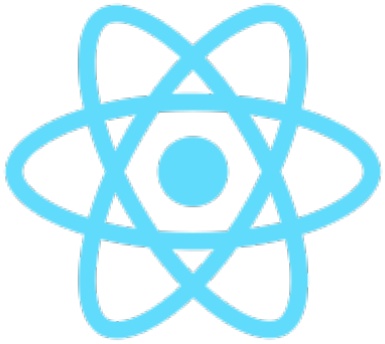

Figura 3.2: Logo de React

## 3.3. Material-UI

**Material-UI** (ver Figura 3.3) es un framework popular para interfaz de usuario o UI (User Interface) en React. Tiene bastantes componentes gráficos ya preparados para React con el objetivo de agilizar la construcción de interfaces gráficas cumpliendo con la normativa de diseño Material Design[2] (ver Figura 3.4).

Figura 3.3: Logo de Material-UI

[2]Material design es una normativa de diseño enfocado en la visualización del sistema operativo Android, además en la web y en cualquier plataforma. Fue desarrollado por Google y anunciado en la conferencia Google I/O celebrada el 25 de junio de 2014.

Figura 3.4: Logo de Material Design

## 3.4. Truffle JS

**Truffle** [19] (ver Figura3.5), es actualmente el framework más popular para el desarrollo de smart contracts en Ethereum, el cual ofrece entre otras cosas:

- Compilación de los smart contracts.
- Automatización de teste de contratos.
- Automatización de los despliegues.
- Gestión de redes Ethereum para desplegar los smart contracts.

Figura 3.5: Logo de Truffle JS

## 3.5. Ganache

**Ganache** [23] ( ver Figura3.6) es una Blockchain personal para el desarrollo rápido de aplicaciones distribuidas (DApps) de Ethereum y Corda. Ganache puede ser utilizado a lo largo de todo el ciclo de desarrollo, lo que permite desarrollar, implementar y probar las DApps en un entorno seguro y determinante.

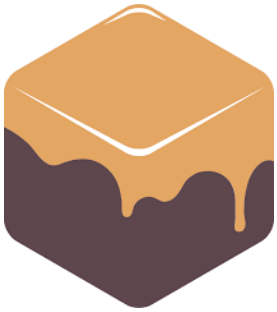

Figura 3.6: Logo de Ganache

## 3.6. MetaMask

La interacción de los usuarios con las DApps de Blockchains como las de Ethereum requieren de un puente, y eso precisamente es lo que hace esta tecnología. **MetaMask** [5], (ver Figura3.7), es una extensión o plugin para navegadores web que permite a los usuarios interactuar de una manera más fácil con las DApps de la Blockchain de Ethereum.

Gracias a esto, los usuarios son capaces de usar dichas DApps de forma más sencilla y todo integrado desde su navegador favorito(soporte con Firefox, Chrome, Opera y Brave).

MetaMask, para conseguir hacer de puente, lo que hace es conectarse a un nodo en concreto de la red de Ethereum llamado INFURA, donde ejecuta los contratos. Para este proyecto, la red a la que se conectará MetaMask será a la que nos cree **Ganache**, permitiendo cambiar entre las 10 cuentas que proporciona. Además de esto, MetaMask permite añadir nuevas cuentas de usuario de la Blockchain a la que esté conectada, simplemente almacenando su clave privada de forma segura. Tras almacenarla, esa cuenta o cuentas añadidas podrán realizar transacciones, además de enviar y recibir Ether con las DApps.


Figura 3.7: Logo de MetaMask

# Capítulo 4

# Aplicación desarrollada

La aplicación desarrollada para el Trabajo Fin de Máster se llama **GuideMe** . En este capítulo se comentan algunas peculiaridades que tiene, para qué sirve y sus diferentes funcionalidades, y también sobre los contratos inteligentes que se han desarrollado para el proyecto.

## 4.1. Peculiaridades

Una de las peculiaridades que tiene la aplicación es el tema de los contratos, pues a pesar de que se tienen cuatro en total, solo se despliega uno en la Blockchain. Este contrato es el que tiene el nombre de **Main.sol**, el cual contiene las funciones que permiten el registro de los usuarios en la aplicación, además de poder añadir transportes también. Los otros contratos sirven como apoyo para que todo esto sea más fácil de manejar. Más adelante se entra más en detalle en cada uno de ellos para que se entienda bien todo.

Otra de las peculiaridades que tiene la aplicación es que es descentralizada (no tiene ningún backend) por lo que toda la información de la aplicación se almacena en la Blockchain. Para que la aplicación funcione lo único que se requiere es tener el navegador y la Blockchain de Ganache inicializada localmente para luego conectarla a MetaMask. Esta conexión se hace de la siguiente forma:

- **Primer paso:** Lo primero que se hace es inicializar, como se dijo anteriormente, la Blockchain con Ganache. Al hacer esto, se abre la interfaz que tiene esta Blockchain con 10 cuentas (cada una de ellas con su dirección y una cantidad de 100 Ether). El Ether que tiene cada cuenta es necesario para pagar el costo que tiene realizar una transacción de los smart contracts.
- **Segundo paso:** Desplegar el contrato en la Blockchain usando Truffle.
- **Tercer paso:** Una vez se tiene la Blockchain inicializada localmente, en la interfaz hay una parte en la que aparecen 12 palabras. Estas palabras son importantes porque permiten a MetaMask conectarse con la Blokchain, solamente hay que copiarlas en un campo que MetaMask tiene asignado para ello y se añade una contraseña elegida por el usuario. Tras confirmar estos campos, ya estaría establecida la conexión.

Tener una Blockchain local (privada) es una ventaja para la aplicación que se desarrolló ya que no se va a tener el problema que tienen las Blockchain públicas frente al almacenamiento masivo de información en ellas. Este problema hace referencia a que cuanta más información se guarde en la Blockchain, el costo es mayor y se penaliza (ya que para almacenar más información hay que crear más bloques, y eso conlleva aplicar potencia computacional extra). Es por ello que, al usar una Blockchain local (privada), el acceso está

restringido y más controlado, por lo que almacenar mayor cantidad de información no supone un problema ya que la creación de bloques es menor que en la Blockchain pública como Ethereum (a la que tiene acceso cualquiera).

Hay que tener en cuenta que tras realizar todo esto, la gestión de las cuentas (es decir, poder cambiar entre una cuenta y otra) lo delega MetaMask. El usuario que esté usando la aplicación en el momento queda determinado por la cuenta que esté seleccionada en MetaMask. Cuando el usuario realiza una acción que conlleve invocar una de las funciones que estén en el contrato y modifiquen los datos (transacción), MetaMask salta haciendo emerger una ventana en el navegador para que el usuario confirme la transacción o no.

## 4.2. Funcionalidades de la aplicación

En esta sección se comentan las funcionalidades que la aplicación tiene, con sus diferentes vistas, que son las siguientes:

- **Inicio:** Esta pestaña muestra la página que se abre nada más arranca la aplicación.
- **Perfil:** En esta pestaña se muestra información del usuario que tenga la cuenta seleccionada en MetaMask (Nombre, apellidos, email, etc.).
- **Registro:** En esta pestaña se lleva a cabo el registro de una cuenta de MetaMask que vaya a ser de usuario al que se le creen las rutas.
- **Ruta:** Esta página muestra la ruta que el usuario tiene que llevar a cabo (los diferentes transportes).
- **Viajar:** Esta pestaña realiza la función de recoger los datos del usuario para crear la ruta.
- **Transportes:** Esta página contiene los transportes que se hayan creado, mostrando la información de los mismos (como el coste o cantidad).
- **Añadir Transportes:** En esta pestaña se lleva a cabo el registro de una cuenta de MetaMask que vaya a ser de transporte para que sea usado en la creación de rutas.
- **Barra de Navegación:** Se le añadió una barra de navegación a la aplicación para tener en ella todas las funcionalidades.

### 4.2.1. Inicio

La página de **Inicio** presenta un texto en el que se introduzca lo que puedes hacer con la aplicación. Es la página de bienvenida de la aplicación que se abre nada más arrancarla y abrirla en el navegador. La Figura 4.1 muestra cómo se vería esta página

### 4.2.2. Barra de Navegación

Esta **Barra de Navegación** contiene todas las demás funcionalidades de la aplicación para tenerlas agrupadas en una única zona. Se muestran u ocultan si el usuario hace click en el botón que hay en ella. En la Figura 4.1 se puede ver el botón en la esquina izquierda superior de la página, y en la Figura 4.2 se puede ver como es esta barra de navegación desplegada.

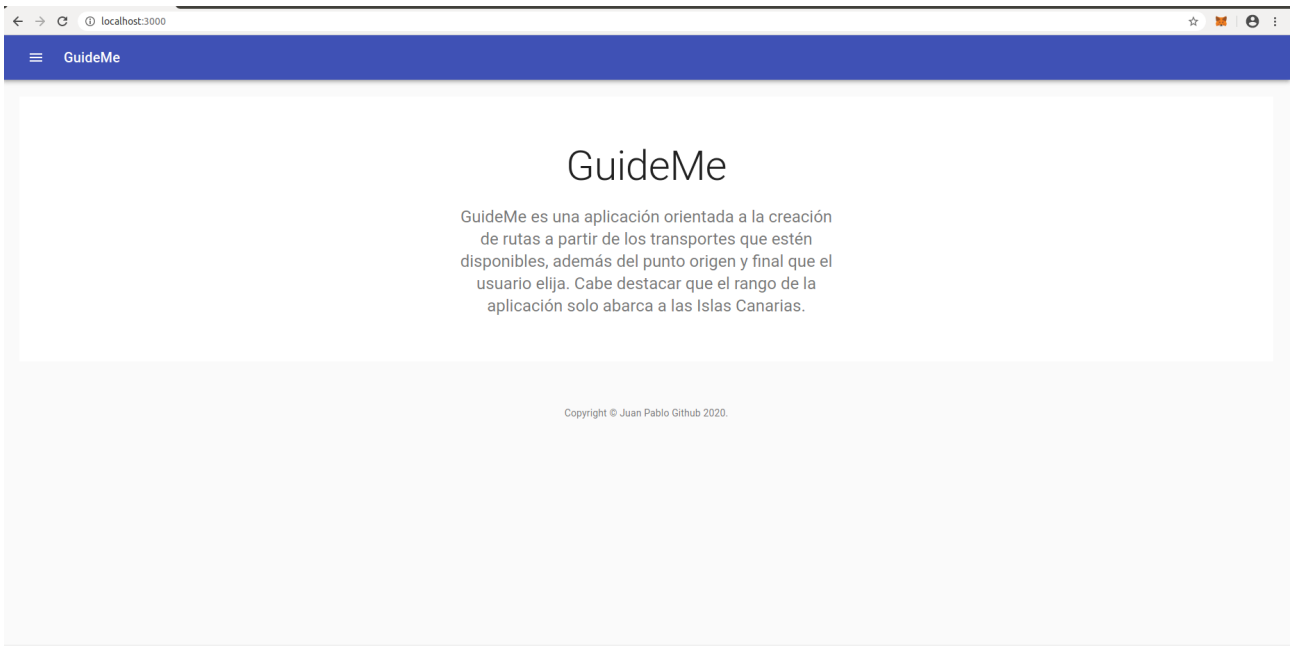


Figura 4.1: Página de Inicio


Figura 4.2: Barra de Navegación

### 4.2.3. Perfil

La página **Perfil** contiene toda la información que se guardó del usuario que se registro con la cuenta de MetaMask activa en el momento. Entre los datos se pueden ver:

- Nombre
- Apellidos
- Email

En la Figura 4.3 se puede ver como luce esta página.

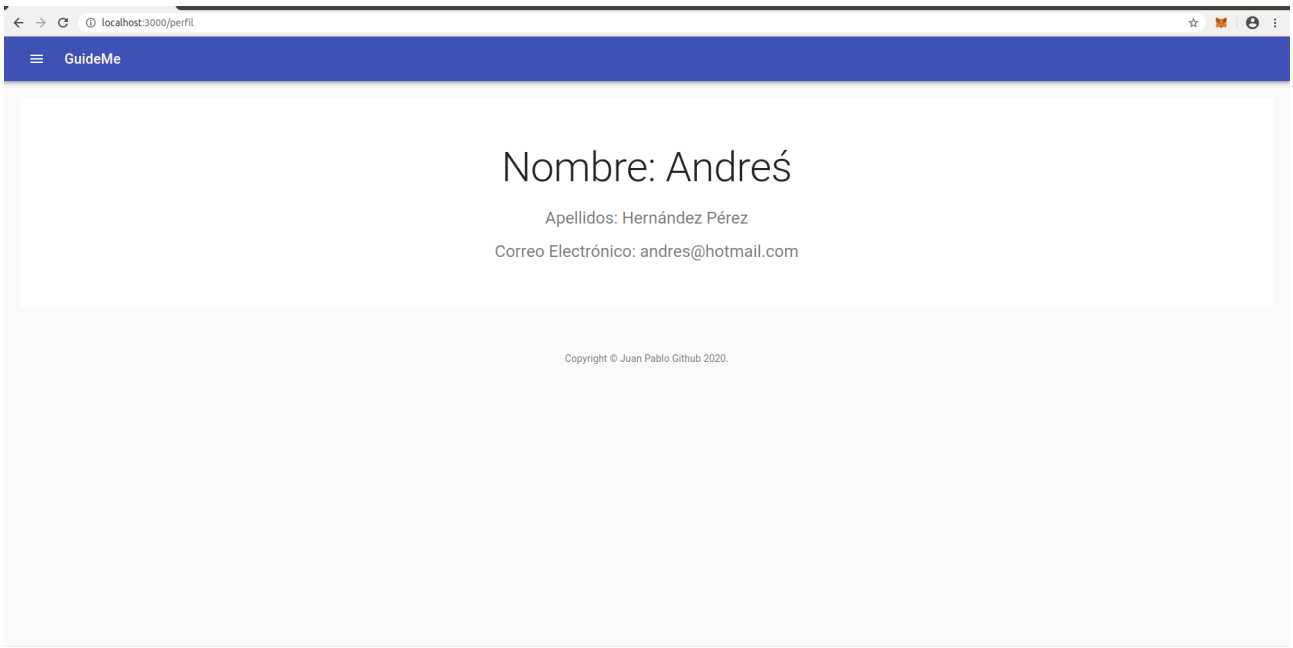


Figura 4.3: Página de Perfil

### 4.2.4. Registro

La página **Registro** se encarga de registrar la cuenta que esté activa en MetaMask como un usuario. Se encarga de recoger los datos del mismo y al pulsar el botón de registrar y confirmar la transacción, esa cuenta queda almacenada como un usuario, para que después cada vez que sea seleccionada en MetaMask, pueda realizar la funcionalidad de la página Viajar. En la Figura 4.4 se puede ver cómo se vería esta página.

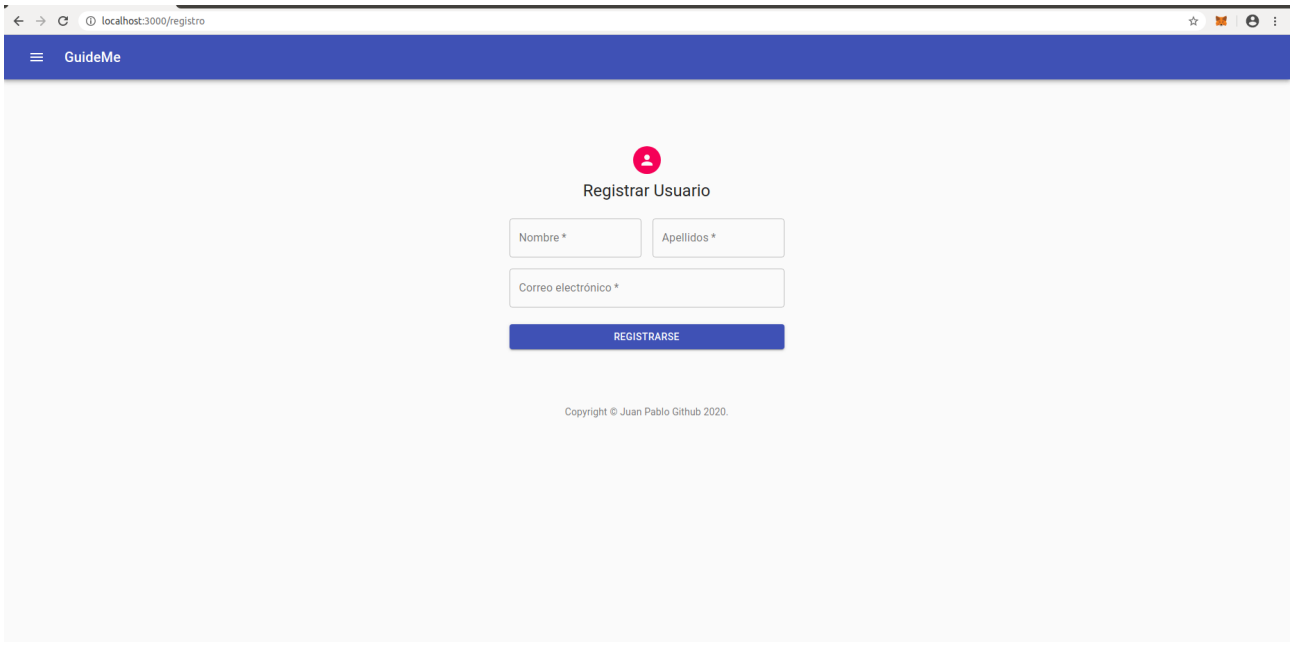


Figura 4.4: Página de Registro

### 4.2.5. Ruta

La página **Ruta** muestra la ruta que el usuario tiene que hacer tras haber completado los pasos de la página Viajar. La ruta se muestra como un Timeline en el que se indique qué transporte tiene que tomar el usuario (mediante un icono), junto a la información que se quiera dar (en este caso se indica la isla donde tiene que tomarlo). En la Figura 4.5 se puede ver el aspecto de la página.

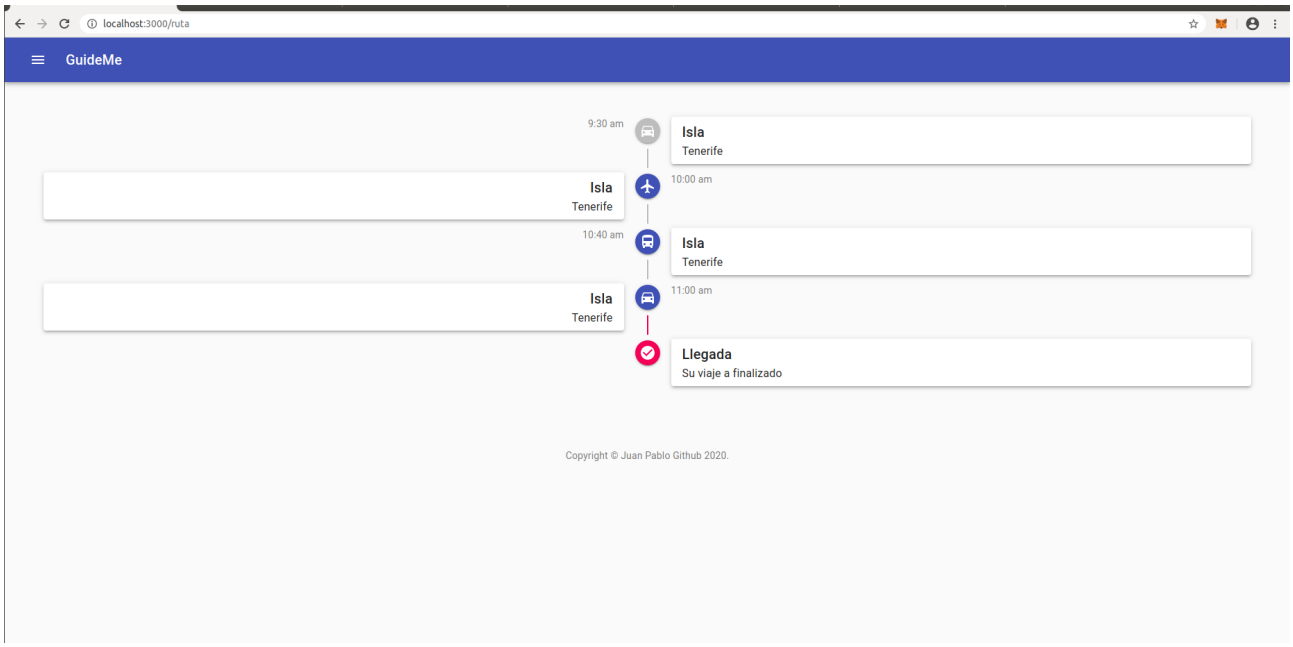


Figura 4.5: Página de Ruta

### 4.2.6. Viajar

En esta página se van a observar varias funcionalidades:

- **Formulario:** En esta parte se van a tomar datos adicionales del usuario que no se tomaron en el registro (teléfono, cuidad, provincia, país) (ver Figura 4.6)
- **Establecer puntos:** En esta parte se va a indicar el punto de origen y fin que el usuario va a establecer para luego crear la ruta según lo que se programó en el contrato (ver Figura 4.7).
- **Resumen de la ruta:** En esta parte se muestra un resumen la ruta creada, indicando los transportes y el coste de cada uno (ver Figura 4.8).
- **Último paso:** En esta parte se muestra un texto de agradecimiento por realizar todos los pasos y haber pagado la ruta establecida (ver Figura 4.9).

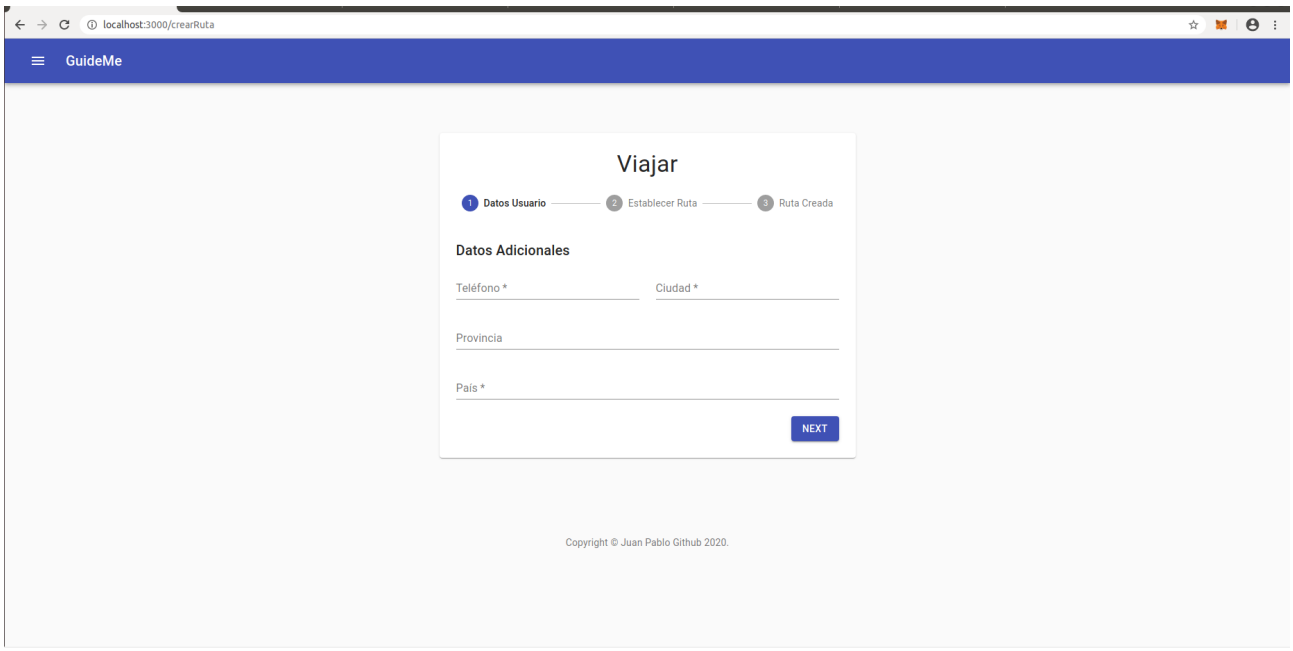


Figura 4.6: Página de Viajar en la parte de Formulario

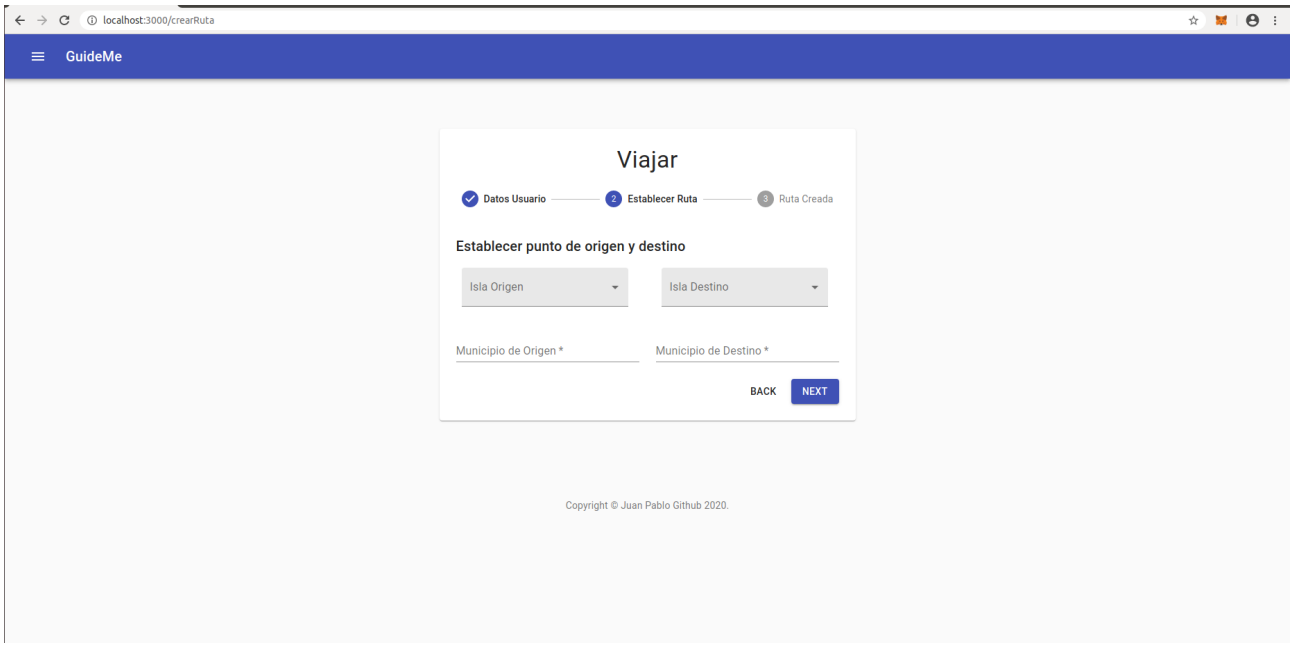


Figura 4.7: Página de Viajar en la parte de Establecer Puntos

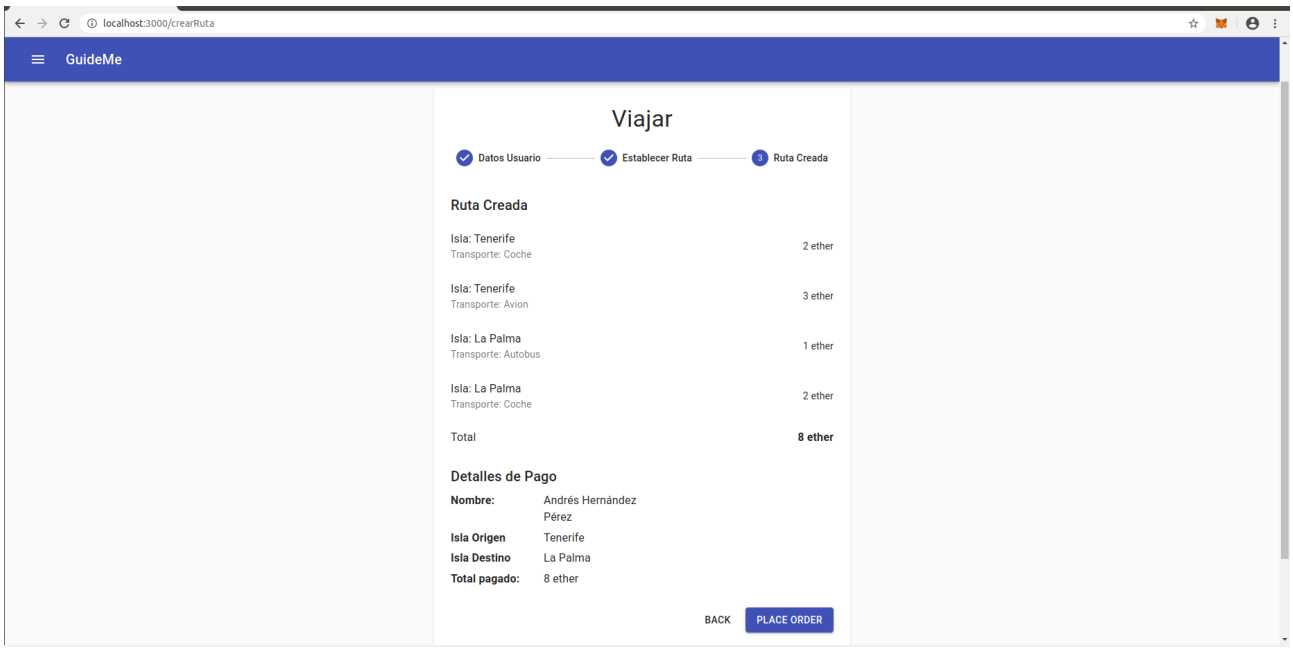


Figura 4.8: Página de Viajar en la parte de Resumen de la ruta

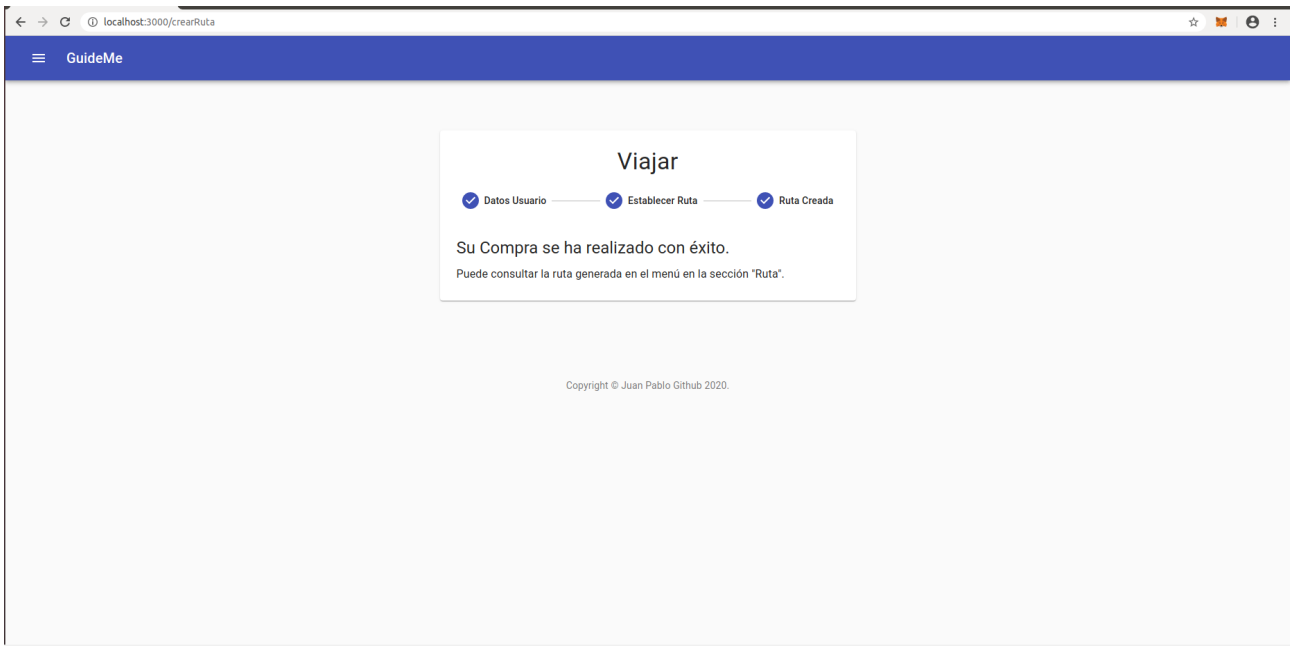


Figura 4.9: Página de Viajar en el Último paso

### 4.2.7. Transportes

La página de **Transportes** va a visualizar todos aquellos transportes que se han creado, junto a su información principal:

- Nombre
- Coste
- Cantidad

A medida que se vayan agregando cuentas que sean de tipo transporte, se han a ir añadiendo en esta página más cajones (uno por cada transporte nuevo) de manera dinámica. Todos estos datos recogidos de la Blockchain.

El aspecto de esta página se puede ver en la Figura 4.10.

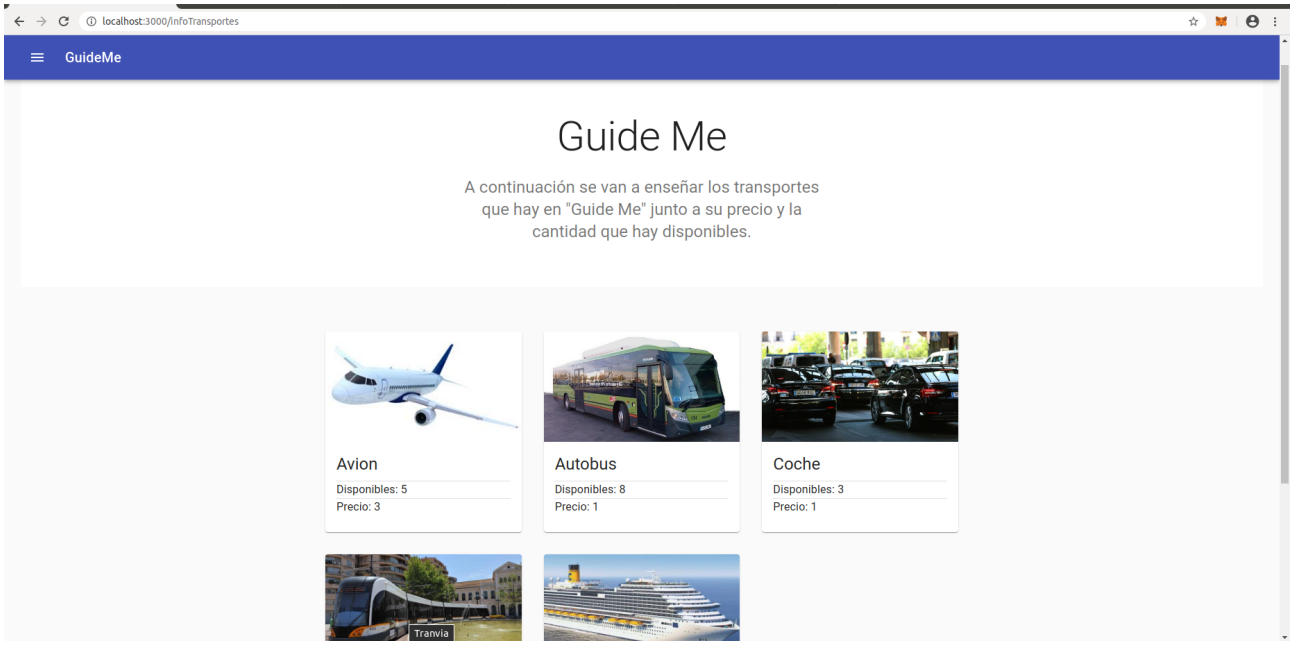


Figura 4.10: Página de Transportes

### 4.2.8. Añadir Transportes

La página **Añadir Transportes** se encarga de registrar la cuenta que esté activa en MetaMask como un transporte. Se encarga de recoger los datos del mismo y al pulsar el botón de añadir y confirmar la transacción, esa cuenta queda almacenada como un transporte. En la Figura 4.11 se puede ver cómo se vería esta página.

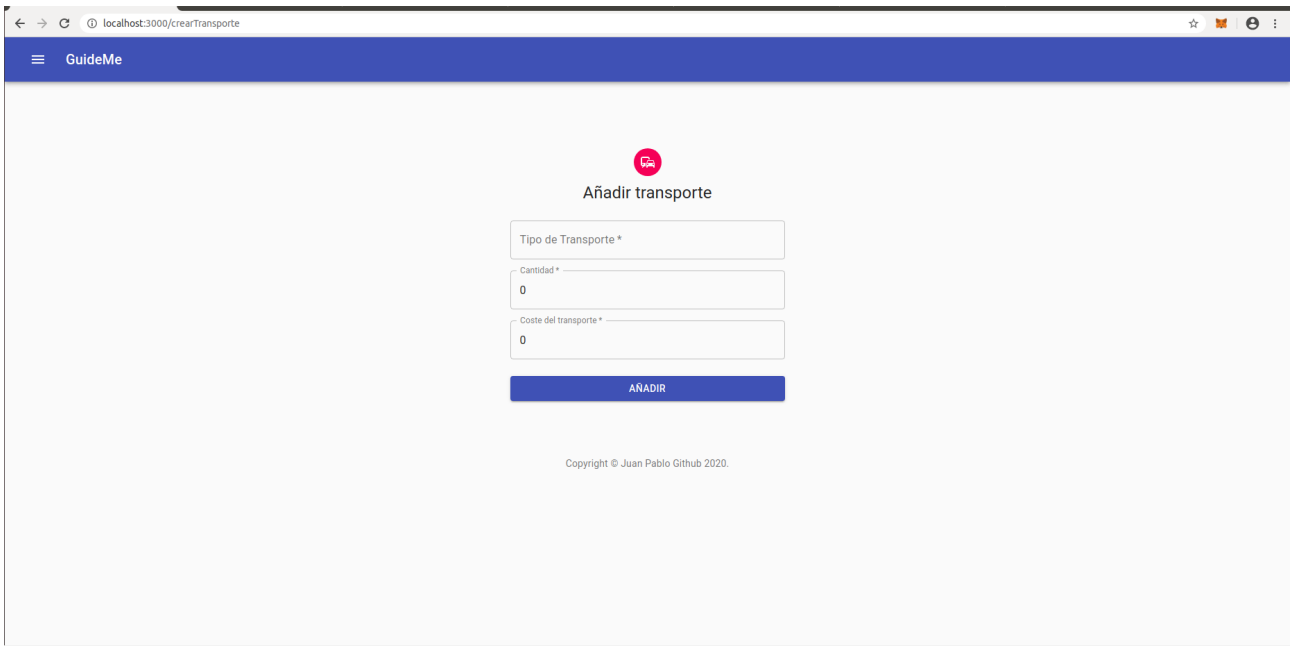


Figura 4.11: Página de Añadir Transportes

## 4.3. Contratos

En esta sección se comentan los diferentes contratos que se han creado para el proyecto. Para cada uno de ellos se muestran los atributos que tienen y se nombran las funciones que tiene cada uno de ellos y el objetivo que tiene la misma. Además, en alguno de ellos se ha realizado un **require** para llamar a los atributos y funciones de otros.

### 4.3.1. Tipo

Este contrato lo que contiene son las declaraciones de los **Structs** que se han creado para:

- **Usuario:** Representa los datos del usuario.
- **Transporte:** Representa los datos del transporte.
- **Ruta:** Representa los datos de la ruta.

Este contrato es incluido en los demás ya que estas estructuras de datos van a ser usadas con frecuencia para poder llevar a cabo el funcionamiento de los métodos de dichos contratos. En la Figura 4.12 se puede ver el cuerpo del mismo.

```
library Tipos {

  struct TransporteDatos {
    address id;
    string nombre;
    uint cantidad;
    uint coste;
  }

  struct RutaDatos {
    string nombreUsuario;
    string nombreTransporte;
    string siguienteTransporte;
    uint coste;
  }

  struct UsuarioDatos {
    address id;
    string nombre;
    string apellidos;
    string email;
    string estado;
  }
}
```

Figura 4.12: Contrato Tipos

### 4.3.2. Transporte

Este contrato contiene la información de un transporte cada vez que sea añadido, permitiendo acceder/modificar a cada uno de sus campos gracias a los getters/setters que se han establecido en el mismo. Este transporte es uno de los **structs** declarados en el contrato Tipos. Los campos que contiene son los siguientes:

- **id:** El id del transporte hace referencia a la dirección de la cuenta que esté activa en MetaMask cuando el transporte sea creado.
- **nombre:** Este campo almacena el nombre del transporte añadido.
- **cantidad:** Este campo guarda la cantidad de vehículos que hay del transporte creado.
- **coste:** Este campo guarda el coste del transporte creado, para consultarlo cuando se vaya a crear la ruta.

En la Figura 4.13 se puede ver el cuerpo del mismo.

```
import './Tipos.sol';

contract Transporte {
  Tipos.TransporteDatos public transporte;

    constructor(string memory _nombre,
                address _id,
                uint _cantidad,
                uint _coste) public {...}

    function getID() public view returns (address) {...}

    function getNombre() public view returns (string memory) {...}

    function getCantidad() public view returns (uint) {...}

    function setCantidad(uint _cantidad) public{...}

    function getCoste() public view returns (uint) {...}

    function getTransporte() public view returns (Tipos.TransporteDatos memory) {...}

}
```

Figura 4.13: Contrato Transporte

### 4.3.3. Usuario

Este contrato va a contener la información de un usuario cada vez que sea añadido, permitiendo acceder/modificar a cada uno de sus campos gracias a los getters/setters que se han establecido en el mismo. Este usuario es uno de los **structs** declarados en el contrato Tipos. Además tiene una variable adicional llamada **rutas** que guarda la ruta del usuario, ya que va almacenando **structs** de rutas con toda su información. Los campos que contiene el usuario son los siguientes:

- **id:** El id del usuario hace referencia a la dirección de la cuenta que esté activa en MetaMask cuando el usuario sea creado.
- **nombre:** Este campo almacena el nombre del transporte añadido.
- **apellidos:** Este campo guarda los apellidos del usuario.
- **email:** Este campo guarda el email del usuario.
- **estado:** Este campo guarda el estado en el que se encuentra el usuario en la ruta, es decir, si está esperando al transporte, en tránsito o ha finalizado el viaje en el transporte que está.

En la Figura 4.14 se puede ver el cuerpo del mismo.

```
import './Tipos.sol';

contract Usuario {
  Tipos.UsuarioDatos public usuario;
  mapping(address => Tipos.RutaDatos[]) public rutas;

    constructor(string memory _nombre,
                string memory _apellidos,
                string memory _email,
                string memory _estado,
                address _id) public {...}

    function getID() public view returns (address) {...}

    function getNombre() public view returns (string memory) {...}

    function getApellidos() public view returns (string memory) {...}

    function getEmail() public view returns (string memory) {...}

    function getEstado() public view returns (string memory) {...}

    function setEstado(string memory _estado) public{...}

    function getUsuario() public view returns (Tipos.UsuarioDatos memory) {...}

}
```

Figura 4.14: Contrato Usuario

### 4.3.4. Main

Este contrato es el principal. Es el que se va a desplegar en la Blockchain con Truffle. Como se puede ver en la Figura 4.16, este contrato llama a los otros tres que se han creado para acceder a sus atributos y métodos. En este contrato se puede observar los siguientes elementos:

- **cuentasCreadas:** Esta variable es un map en el que se almacena su dirección (identificador) y un booleano (valor) indicando si la cuenta que está activa en MetaMask en el momento (la que va a ejecutar alguna de las funciones) es una cuenta creada (true) o no (false).
- **transportesCreados:** Al igual que los usuarios, este map indica las direcciones que hay creadas de tipo transporte.
- **transportes:** Es un map que almacena las direcciones de las cuentas que se crearon como transporte y un objeto Transporte, para que si la cuenta que está activa en el momento en MetaMask es un transporte, acceder a sus valores si así se requiere.
- **usuarios:** Al igual que el map anterior, esta variable almacena las direcciones de las cuentas que se crearon como usuario y un objeto Usuario, para que si la cuenta que

está activa en el momento en MetaMask es un usuario, acceder a sus valores si así se requiere (como por ejemplo visualizar sus datos en la página perfil).

- **transportesArray:** Esta variable es un array de structs de transportes, para que a medida que se van creando transportes, se van almacenando en esta variable, y es más cómodo acceder a todos los transporte creados y sus valores, para poder visualizarlos en la página Transportes.
- **numTransportes:** Esta variable va a contener el número de transportes creados.
- **crearTransporte:** Esta función es la que se llama cada vez que se cree un transporte, para añadirlo a la Blockchain.
- **getUsuario:** Esta función devuelve un Usuario cuya dirección sea id.
- **crearUsuario:** Esta función es la que se llama cada vez que se cree un usuario, para añadirlo a la Blockchain.
- **crearRuta:** Esta función es la que se encarga de crear la ruta del usuario con dirección id según los datos que se manden de la página Viajar. Para crear la ruta se mira primero si la isla origen y destino son la misma (para omitir o no el avión y barco). Una vez se haya pasado esta parte, se va creando la ruta de manera aleatoria pero fijándonos en la cantidad de transporte de cada tipo que hay disponible y en el coste. Primero se va cogiendo aquellos que tengan menor costes, y si no hay disponibles, se pasa al siguiente.

Todos estos contratos están escritos en el lenguaje **Solidity** (ver Figura 4.15).

Figura 4.15: Solidity

```
import './Tipos.sol';
import './Transporte.sol';
import './Usuario.sol';

contract Main {
    mapping(address => bool) public cuentasCreadas;
    mapping(address => bool) public transportesCreados;
    mapping(address => Transporte) public transportes;
    mapping(address => Usuario) public usuarios;

    Tipos.TransporteDatos[] public transportesArray;

    uint public numTransportes;

    constructor() public {...}

    function crearTransporte(string memory _nombre,
                             address _id,
                             uint _cantidad,
                             uint _coste) public {...}

    function getUsuario(address id) public view returns (Tipos.UsuarioDatos memory) {...}

    function crearUsuario(string memory _nombre,
                          string memory _apellidos,
                          string memory _email) public {...}

    function crearRuta(address _address) public {...}
```

Figura 4.16: Contrato Main

# Capítulo 5

# Conclusiones y líneas futuras

En este capítulo se presentan algunas de las conclusiones que se han sacado durante el desarrollo de este trabajo, además de varias posibles líneas futuras de desarrollo.

## 5.1. Conclusiones

Al realizar el proyecto usando la tecnología Blockchain, con su estudio previo, no sólo se ha llegado a la conclusión de que tiene muchas posibilidades de uso a futuro (ya que tiene bastante potencial), sino que además proporciona ventajas solamente al aplicarla en algunas aplicaciones concretas.

Además de todo esto, haber realizado una DApp usando esta tecnología ha servido para asentar y comprender cómo funciona, tanto para saber cómo se lleva a cabo una transacción (es decir, conocer el conjunto de pasos que tiene que dar para poder ser ejecutada), como la criptografía que la rodea (cifrado asimétrico, funciones hash, etc.), poniendo en práctica además todo lo aprendido durante el curso (ya que estos conceptos han sido nombrados en el periodo de estudio).

Entre las ventajas que tiene se encuentran las siguientes:

- Es una tecnología que podría considerarse **incorruptible**, ya que al tener distintas formas de verificación de datos, la alteración de los mismos por parte de terceros es difícil o prácticamente imposible. Cada transacción es inmutable y no puede ser eliminada o modificada sin que se detecte.
- Tiene **transparencia**, es decir, aunque cualquier movimiento que se haga no puede ser modificado por nada ni por nadie, puede ser vista públicamente por cada parte.
- Es un proceso inmediato porque, al no existir intermediarios, el sistema informático utilizado puede estar operativo todo el tiempo.
- Al ser un sistema descentralizado, la fiabilidad que tiene es alta, ya que si un nodo de la cadena cae, la Blockchain seguirá funcionando con que sólo haya uno activo.

Por todas estas ventajas y más es por lo que Blockchain da seguridad a las aplicaciones que la utilizan.

## 5.2. Líneas futuras

Entre las principales líneas futuras que se pueden contemplar tanto para el rendimiento de la aplicación como para mejoras en cuanto a la interfaz destacan las siguientes:

- La primera mejora que se viene a la cabeza es en todo el tema del almacenamiento. Habría que buscar alguna alternativa para el almacenamiento distribuido de datos, para que así no se vea afectado el rendimiento de la Blockchain con tantos datos que se van almacenando a lo largo del tiempo en ella.
- Otra posible mejora consiste en añadir en la página para establecer un viaje, un mapa que permita marcar tanto el punto de origen como el punto destino de la ruta, y así darle más comodidad a los usuarios, ya que al tener selectores y entradas para escribir esos datos, si alguna vez se quieren añadir nuevos puntos de origen y destino, habría que añadirlos manualmente. Con el mapa, por otro lado, si se consigue hacer funcionar usando las coordenadas para marcar los puntos, no habría que añadir nada manualmente en el código.
- Una tercera mejora sería ampliar el alcance de la aplicación a nivel nacional y no sólo en las islas, añadiendo más tipos de transportes (como el metro) y más puntos origen/destino.

# Capítulo 6

# Summary and conclusions

This chapter presents some conclusions that have been drawn during the development of this work, and some future lines.

## 6.1. Conclusions

During the development of the project using Blockchain, with its previus study, it has been concluded that it has many possibilities for future use (cause it has quite potential), in addition to providing advantages only by using it in some application that is being developed.

In addition to all this, having made a DApp using this technology led us to consolidate and understand how it works, both to know how a transaction is carried out (the group of steps it has to be executed) and the cryptography that surrounds it (asymmetric encryption, hash functions, etc.), also putting into practice everything learned during the course (since these concepts have been named in the period of study).

We can appreciate the following advantages:

- It is a technology that could be considered **incorruptible**, since having different ways of data verification, the alteration of data by third parties is very difficult or impossible. Each transaction is immutable and cannot be eliminated or modified without detection.
- It is **transparent**, although any movement made cannot be modified by anything or anyone, it can be seen publicly by each party.
- It is an inmediate process because there are no intermediaries, so the computer system used can be operational all the time.
- Since Blockchain is a decentralized system, its reliability is high, because if a node of the chain falls, the Blockchain will continue to work with only one active node.

For all these advantages and more is why Blockchain gives security to the applications that use it.

## 6.2. Future work

Below some future lines that can be foreseen both for the performance of the application and for interface improvements are:

- The first improvement that comes to mind is the whole issue of storage. It would be necessary to look for some alternative for the distributed storage of data, so that the performance of the Blockchain is not affected with so much data being stored in it over time.
- Another possible improvement consists in adding, in the page to establish a trip, a map that allows to mark both the origin and destination point of the route, in order to give more comfort to the users, since having selectors and entries to write those data, if you ever want to add new origin and destination points, you would have to add them manually in the code. On the other hand, with the map, if you can get it to work using the coordinates to mark the points, you won't have to add anything manually.
- A third improvement would be to extend the scope at national level (not only on the islands), adding more types of transport (such as metro) and more origin/destination points.

# Capítulo 7

# Presupuesto

En este capítulo se presentan los costes estimados del proyecto, distinguiéndose los recursos necesarios a nivel de hardware, software y de personal.

## 7.1. Costes de software

| Software | Coste |
|---|---|
| Framework Truffle | 0 € |
| Framework React | 0 € |
| Framework Material-UI no premium | 0 € |
| Ganache | 0 € |
| MetaMask | 0 € |
| **Total** | o € |

Cuadro 7.1: Tabla costes software

## 7.2. Costes de hardware

| Software | Coste |
|---|---|
| Ordenador portátil | 580 € |
| **Total** | 580 € |

Cuadro 7.2: Tabla costes hardware

## 7.3. Costes del personal

En esta sección se exponen los costes de los recursos humanos necesarios para el desarrollo de la aplicación. Se ha tomado como como referencia una jornada laboral de 4 horas durante 5 días cada semana, y el precio por hora de trabajo está en torno a unos 18 €:

| Software | Jornadas | Coste |
|---|---|---|
| Análisis de las tecnologías a utilizar | 8 | 576 € |
| Estudio del funcionamiento de la Blockchain, MetaMask y Ganache | 9 | 648 € |
| Formación en el framework React, contratos inteligentes y desarrollo de la aplicación | 15 | 1080 € |
| Aplicación del framework Material-UI | 3 | 216 € |
| Detección y corrección de errores | 5 | 360 € |
| **Total** | 40 | 2880 € |

Cuadro 7.3: Tabla costes de recursos humanos

## 7.4. Costes totales

| Software | Coste |
|---|---|
| Coste software | 0 € |
| Coste hardware | 580 € |
| Coste RRHH | 2880 € |
| **Total** | 3460 € |

Cuadro 7.4: Tabla costes totales

# Apéndice A

# Anexo

## A.1. Diagrama de caso de uso

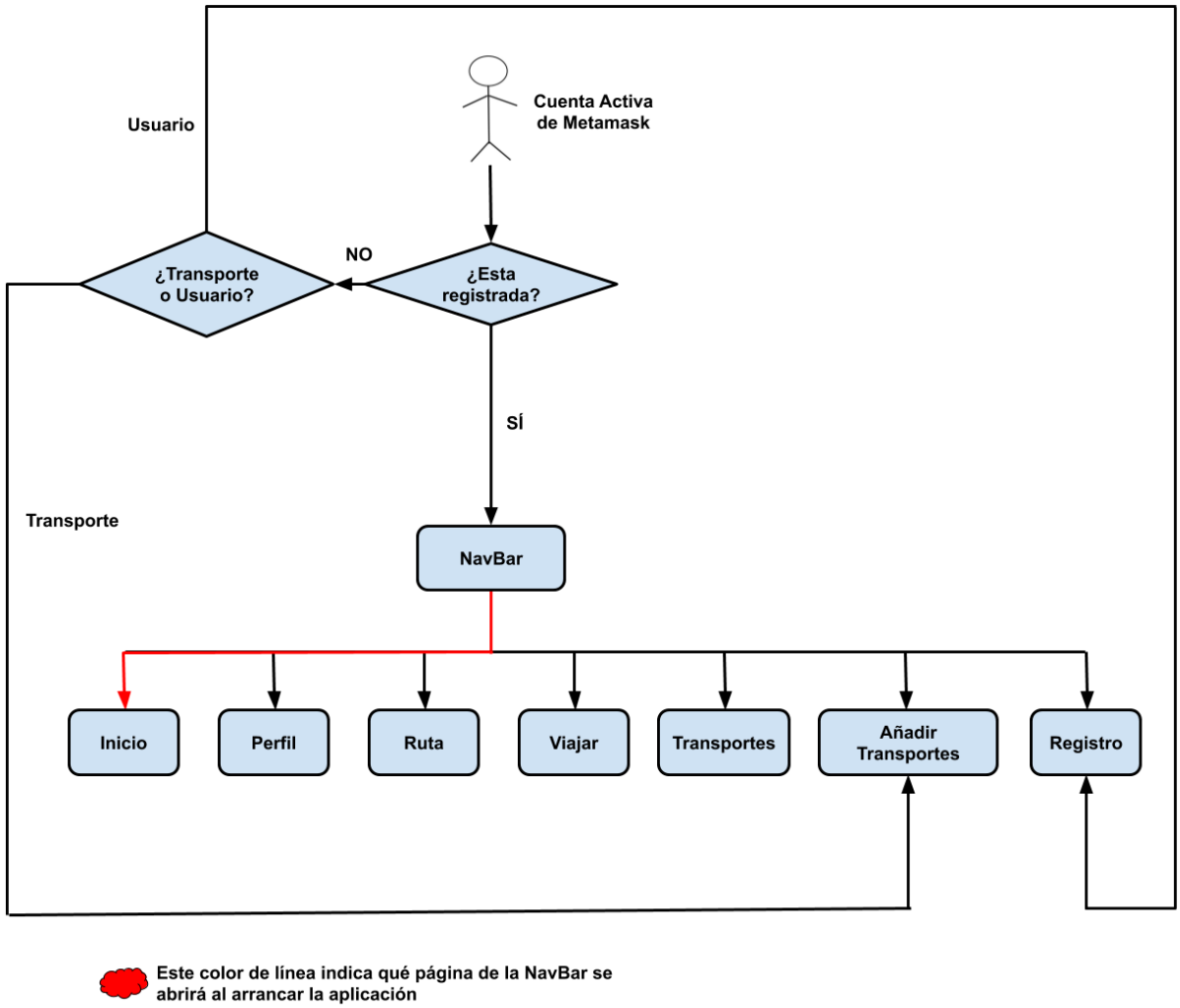


Figura A.1: Diagrama de caso de uso de la aplicación

## A.2. Estructura de ficheros

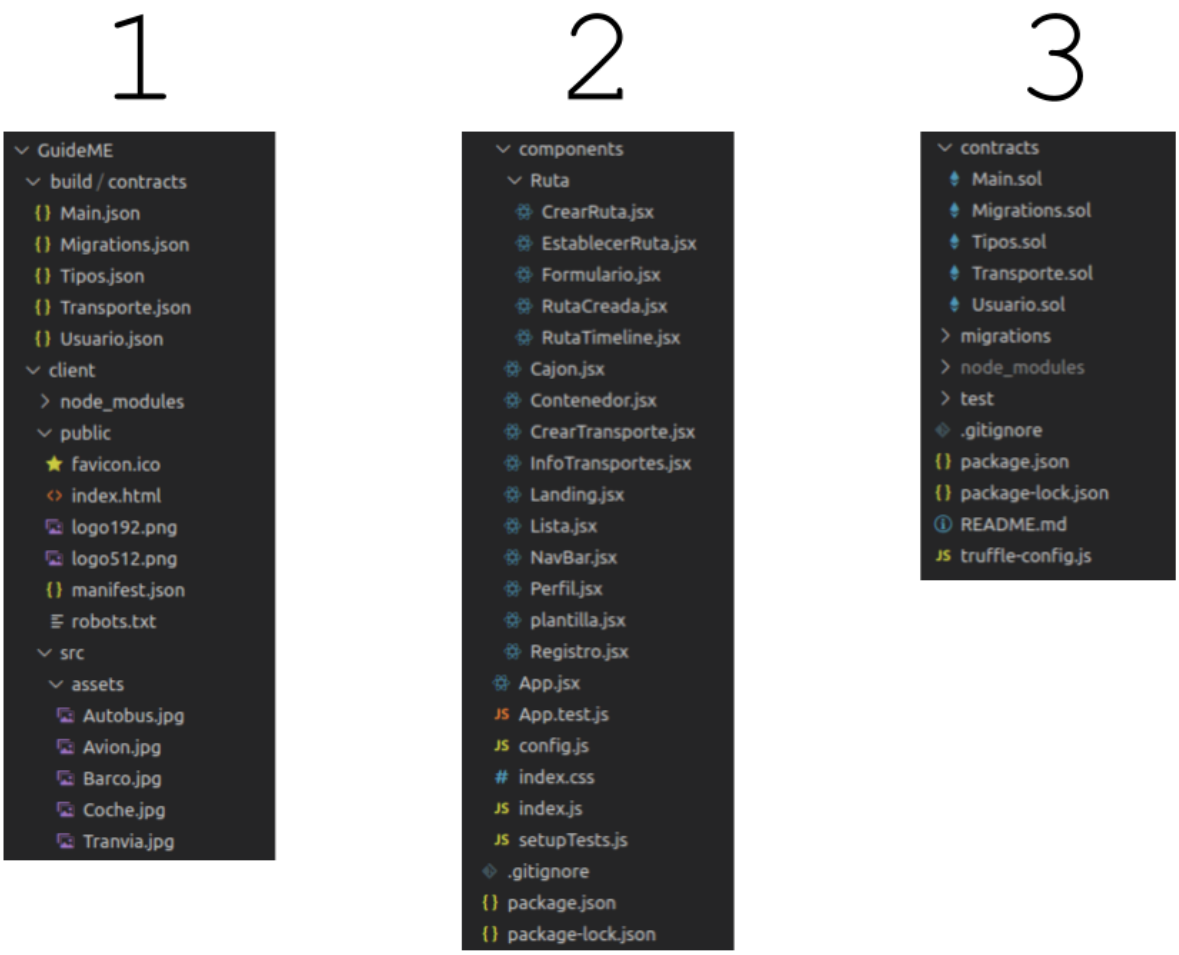


Figura A.2: Estructura de ficheros

## A.3. Esquema de Ruta

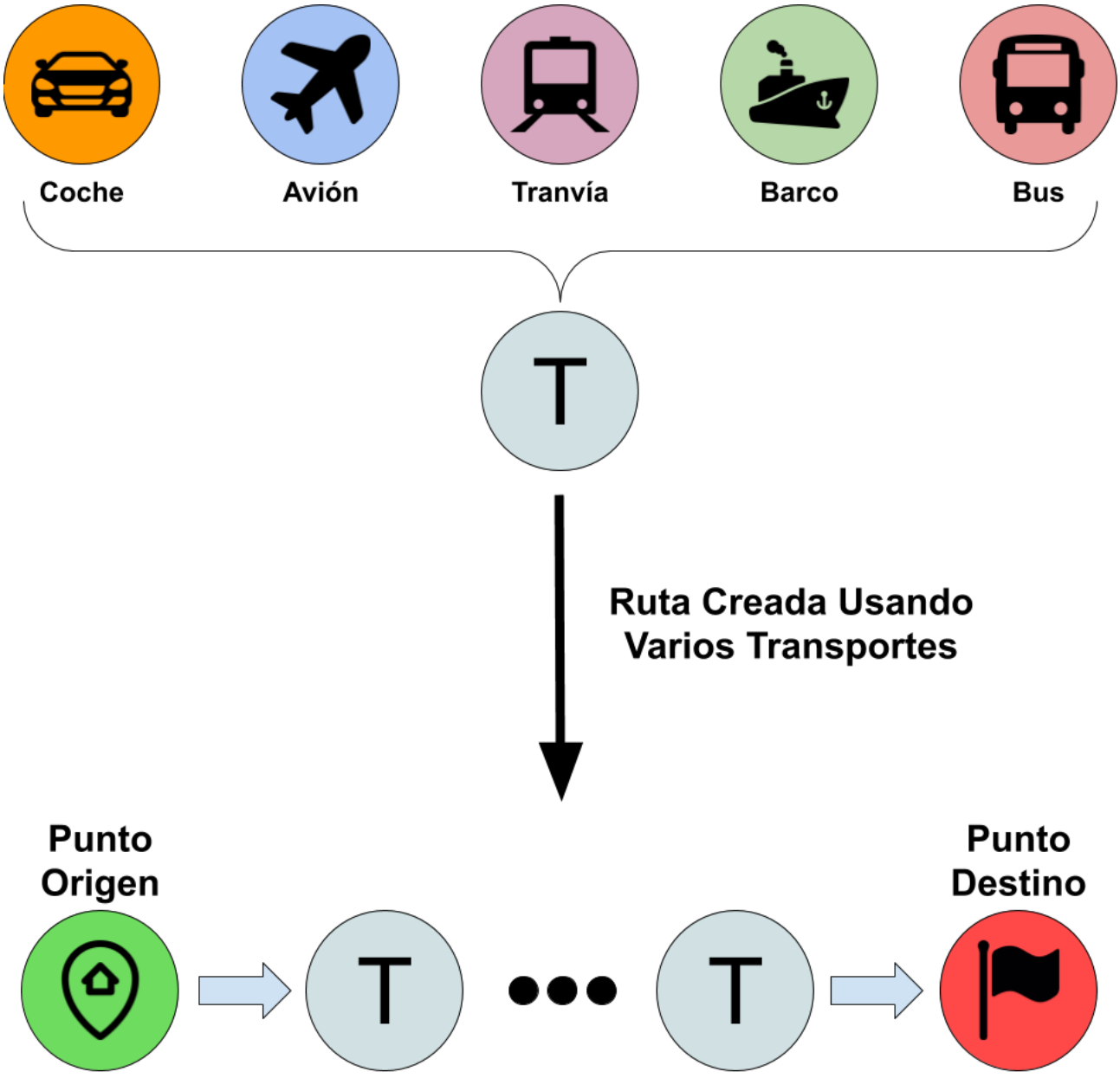


Figura A.3: Cadena de transporte multimodal simplificada

# Bibliografía


[1] Binance Academy. La historia de blockchain. https://academy.binance.com/es/blockchain/history-of-blockchain. Accessed: 2020-06-23.

[2] Binance Academy. ¿qué es un algoritmo de consenso blockchain? https://academy.binance.com/es/blockchain/what-is-a-blockchain-consensus-algorithm. Accessed: 2020-06-24.

[3] Bit2me Academy. Cuántos tipos de blockchain hay. https://academy.bit2me.com/cuantos-tipos-de-blockchain-hay/. Accessed: 2020-06-24.

[4] Bit2me Academy. Qué es la criptografía asimétrica. https://academy.bit2me.com/que-es-hash/. Accessed: 2020-06-24.

[5] Bit2me Academy. Qué es metamask – la forma más fácil de usar dapps. https://academy.bit2me.com/que-es-metamask-la-forma-mas-facil-de-usar-dapps/. Accessed: 2020-06-28.

[6] Bit2me Academy. Qué es poa (proof of authority – prueba de autoridad). https://academy.bit2me.com/que-es-proof-of-authority-poa/. Accessed: 2020-06-24.

[7] Bit2me Academy. Qué es prueba de trabajo / proof of work (pow). https://academy.bit2me.com/que-es-proof-of-work-pow/. Accessed: 2020-06-24.

[8] Bit2me Academy. Qué es un hash. https://academy.bit2me.com/que-es-hash/. Accessed: 2020-06-24.

[9] Miguel Angel Alvarez. Qué es react. por qué usar react. https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html. Accessed: 2020-06-28.

[10] Yukiteru Amano. Árboles de merkle: qué son y sus distintos usos en la actualidad. https://agorachain.org/arboles-merkle-usos/. Accessed: 2020-06-24.

[11] Criptotario. Prueba de trabajo vs prueba de participación. https://criptotario.com/prueba-de-trabajo-vs-prueba-de-participacion. Accessed: 2020-06-24.

[12] Carla Ferrer. El smart contract o contrato inteligente. https://protecciondatos-lopd.com/empresas/smart-contract/#:~:text=Origen%20del%20contrato%20inteligente,cript%C3%B3grafo%20en%20el%20a%C3%B1o%201994. Accessed: 2020-06-23.

[13] GuiBit. Qué es un nonce. https://guiabit.win/que-es-un-nonce/. Accessed: 2020-06-23.

[14] HCXI. Blockchain insurance and healthcare. https://www.hcxi.co/. Accessed: 2020-06-23.

[15] HYPERLEDGER. Hyperledger indy. https://www.hyperledger.org/use/hyperledger-indy. Accessed: 2020-06-23.

[16] IBM. Ibm food trust. a new era for the world's food supply. https://www.ibm.com/blockchain/solutions/food-trust. Accessed: 2020-06-23.

[17] IG. ¿qué es ethereum y cómo funciona? https://www.ig.com/es/ethereum-trading/que-es-ether-y-como-funciona. Accessed: 2020-06-24.

[18] iSolve. isolve. https://isolve.io/. Accessed: 2020-06-23.

[19] Paula Pousa Martínez. Aprende a usar truffle. http://catedraiecisa.etsiinf.upm.es/aprende-a-usar-truffle/#:~:text=Truffle%20es%20actualmente%20el%20framework,Automatizaci%C3%B3n%20de%20los%20despliegues. Accessed: 2020-06-28.

[20] Miethereum. Cómo funciona blockchain y sus partes. https://www.miethereum.com/blockchain/como-funciona/. Accessed: 2020-06-23.

[21] Cecilia Pastorino. Blockchain: qué es, cómo funciona y cómo se está usando en el mercado. https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/. Accessed: 2020-06-23.

[22] patientoryinc. Patientory. https://patientory.com/. Accessed: 2020-06-23.

[23] Truffle Suite. Ganache overview. https://www.trufflesuite.com/docs/ganache/overview. Accessed: 2020-06-28.

[24] Agora Technologies. Brinding voting systems into the digital age. https://www.agora.vote/. Accessed: 2020-06-23.

[25] Cercle TIC. Aplicaciones reales de blockchain. http://cercledirectorstic.com/2017/11/29/aplicaciones-reales-blockchain/. Accessed: 2020-06-23.

[26] MDN web docs. Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript. Accessed: 2020-06-28.