UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UN SERVICIO DE SEGUIMIENTO DE MERCANCÍAS BASADO EN LA CADENA DE BLOQUES ‘ETHEREUM’

JAIME DE FRUTOS CEREZO

2019

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Desarrollo de un servicio de seguimiento de mercancías basado en la cadena de bloques 'Ethereum'.

Autor: D. Jaime de Frutos Cerezo.

Tutor: D. Santiago Pavón Gómez.

Ponente: D. …….

Departamento: Departamento de Ingeniería de Sistemas Telemáticos.

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UN SERVICIO DE SEGUIMIENTO DE MERCANCÍAS BASADO EN LA CADENA DE BLOQUES 'ETHEREUM'

JAIME DE FRUTOS CEREZO

2019

RESUMEN

Blockchain o la cadena de bloques es una tecnología emergente muy novedosa y que ha supuesto una gran revolución. Esta cadena de bloques puede almacenar de forma distribuida y muy segura multitud de datos, ya sean transacciones monetarias, contratos inteligentes o hasta la información de la ruta que ha seguido una mercancía.

En este TFG se quiere hacer hincapié en esta última aplicación. En la cadena de bloques se podría reflejar numerosos datos de todo tipo de mercancías: lugares por los que ha pasado(trazabilidad), fechas de producción, fechas de caducidad… Cualquiera podría acceder a esta información, pero no modificarla, aquí reside su gran fortaleza y por lo que actualmente, muchas grandes corporativas están contando con esta tecnología para ofrecer más fiabilidad a sus clientes.

Con este trabajo se busca implementar esta aplicación de trazabilidad de mercancías mediante el uso de Blockchain contando con la red 'Ethereum', la cual ofrece buenas características como gran fiabilidad, ningún tipo de censura, fraudes o acciones de terceras partes. A su vez se desarrollaría una aplicación web para el uso de esta implementación.

El uso que se le daría a este servicio sería el siguiente:

- Los involucrados en la creación, transporte, venta… de las mercancías podrían subir a nuestra Blockchain datos de éstas cada vez que pasan por sus manos. Por ejemplo, cuando un productor acaba su tarea con su producto, éste subiría al primer bloque de la cadena las especificaciones del producto, fechas importantes a tener en cuenta y otros datos útiles; el transportista, subiría datos de la ruta que ha seguido el producto en el mismo bloque… y así hasta llegar al vendedor y a los clientes. Un único producto abarcaría un bloque de la cadena. Esta utilidad de recolección de información estaría abstraída en una aplicación web, por ejemplo, así los encargados de emplear este servicio no necesitarían especializarse en esta tecnología de reciente uso. Hoy en día hay muy pocas personas que se hayan dedicado a estudiar Blockchain, la mayoría de sus usuarios son autodidactas que se interesaron en el tema por pura afición. (Cabe destacar que todos estos datos irían firmados digitalmente para aportar mayor integridad y autenticidad a la cadena)

- Por otra parte, los clientes dispondrían de una forma sencilla de acceder a los datos de la cadena. Por ejemplo, el vendedor les ofrecería un código QR en cada etiqueta de producto donde con un simple escaneo el cliente obtendría todos los datos del producto de una forma rápida. (De forma similar se podría realizar con un link en una web o una aplicación móvil especializada con una lista de productos)

Este TFG se encargaría del desarrollo de todas las aplicaciones envueltas en los procesos anteriores, serían aplicaciones web para registrar productos y su información y la aplicación necesaria para la utilización de los supuestos clientes.

SUMMARY

Blockchain is one of the newest technologies and it has supposed a revolution. This blockchain can save in a distributed and safe way many data types, like monetary transactions, smart contracts or even information about the route that a ware has traversed.

This Final Degree Essay wants to focus on this last application. It could be possible to save much information about wares in the blockchain: places it has go through (traceability), production dates, expiration dates... Anyone could reach this information but not everyone could change it. Here lies its strength and that is why many great corporations are starting to use this technology to offer more reliability to their clients.

This essay seeks to implement this application using the well-known Blockchain network "Ethereum". This network assures great reliability, no censure, frauds or third part actions. At the same time, a web application would be developed to implement all these features.

This application would count with the following services:

On one hand, companies involved in the creation, transport, selling... of goods could save in our Blockchain data of theses goods whenever they manipulate them. For example, when a producer ends their duty with his product, he would upload, to the first block of the network, many specifications, important dates and more; the shipper would upload the route information it has gone through... and so on until arriving to the seller and client. One block of the chain could cover one product. This information recollection utility would be abstracted in a web application so the ones using this app, clients and manipulators, will not need to be experts using this technology. Nowadays, there are few people who has studied Blockchain technology, most of the people who works with it are self-taught, that were interested in it. (It is important to highlight that every data would be digitally signed to contribute more integrity and authenticity to the chain).

On the other hand, clients could easily access to the chain information. For example, sellers could offer a QR code attached to the product so the client could scan it with his/her phone camera and look up every piece of information. (It could also be done by clicking on a link in a web or through a specialised phone app that has an item list).

This Final Degree Essay will cover every development needed to create the applications involved in the processes mentioned above: A web application to register products and another one to check the information of these products to be used by clients.

PALABRAS CLAVE

Cadena de bloques, contrato inteligente, aplicación web, Ethereum, trazabilidad, solidity, javascript, html, css, react, drizzle, truffle, Amazon Web Services, AWS.

KEYWORDS

Blockchain, smart contract, web application, Ethereum, trazability, solidity, javascript, html, css, react, drizzle, truffle, Amazon Web Services, AWS.

ÍNDICE DEL CONTENIDO

1. INTRODUCCIÓN

1.1. MOTIVACIONES

Actualmente, nos encontramos en la era de la información, todo el mundo desea tener más conocimientos y de mayores calidades. Alrededor de esta idea, surgen tecnologías nuevas diariamente. Una de estas tecnologías más recientes es 'Blockchain'. Con ella nos podemos asegurar que la información manejada por esta tecnología no está comprometida.

El desarrollo de nuevas tecnologías conlleva la aparición de nuevas formas de hackeo. Pero para la cadena de bloques no es un problema, ya que es una de las tecnologías más seguras hoy en día. Blockchain es lo que sugiere su traducción, una cadena de bloques. Estos bloques representan una unidad de información. Pueden contener desde programas hasta transacciones monetarias y toda esta información está acompañada de "hashes" [1]. Estos hashes actúan como firma o garantía de que la información no se ha modificado en el propio bloque y en los anteriores. Gracias a este atractivo, Blockchain es una de las tecnologías favoritas para aportar autenticidad e integridad a los sistemas software.

Por otra parte, al ser una tecnología tan novedosa, no hay muchos profesionales que sepan manejarla. Este motivo y el propio atractivo de la tecnología me han llevado a elegirla para mi Trabajo Fin de Grado.

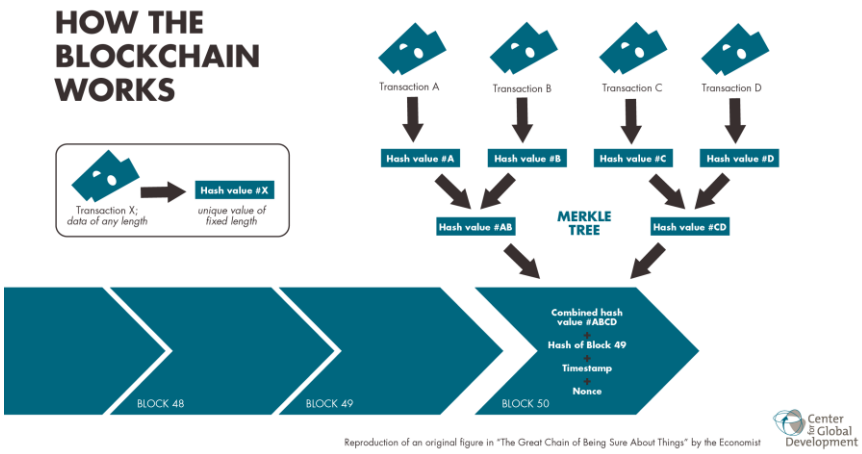

Ilustración 1: Flujo explicativo de la tecnología Blockchain

1.2. OBJETIVOS

Esta es la lista de objetivos a cumplir en este TFG:

- Estudiar la tecnología Blockchain, sus funcionalidades, características y fortalezas.
- Estudiar la solución que nos propone Ethereum para el despliegue de DApps o aplicaciones distribuidas.
- Estudiar la metodología de desarrollo de aplicaciones con las tecnologías descritas en "Estado del Arte".
- Aplicar todos los conocimientos a un caso de estudio concreto: La realización de una aplicación Web para el seguimiento de las mercancías a lo largo de los procesos de producción y distribución.

Al final de la memoria, se recorrerá esta lista para comprobar si se han conseguido cumplir.

2. ESTADO DEL ARTE

Desde la antigüedad, la gente ha estado buscando formas de evitar que sus comunicaciones se viesen comprometidas: desde los sobres cerrados con cera y sello único para asegurar la integridad del contenido, hasta la actualidad, con el cifrado de gran parte de las comunicaciones digitales. Hoy en día, no hay conversación que se realice a distancia que no esté cifrada, aquí se incluyen incluso las conversaciones telefónicas.

Por otra parte, no solo hay que asegurar las comunicaciones, también hay que asegurar el guardado de información. Las grandes compañías poseen bases de datos con cuidadosos controles y varias copias de seguridad. Para que, en el peor de los casos, se pueda recuperar la valiosa información sin perder tiempo y/o dinero.

En este TFG, se usarán numerosas tecnologías para poder asegurar nuestra información, ya se ha hablado un poco de una de ellas: Blockchain o cadena de bloques. A continuación, se aportará un poco más de información acerca de ella y demás tecnologías usadas.

2.1. RED DE 'ETHEREUM'

En la actualidad, existen varias redes de Blockchain, como pueden ser BitCoin [2], Ethereum [3], Hyperledger [4]… No obstante, no todas son iguales u ofrecen lo mismo. BitCoin está orientada o dedicada a su criptomoneda y a transferencias. Ethereum, nuestra opción, permite el uso de Smart Contracts [5] en su red, lo que nos permite desplegar código ejecutable sobre la Blockchain.

Por otra parte, existen varios tipos de clasificaciones para las distintas redes [6]:

- Según el acceso a los datos: Existen redes públicas (BitCoin) y privadas. Ethereum permite crear redes privadas y también posee la suya propia pública.
- Según los permisos: Existen redes permisionadas, en las que el procesado de las transacciones está acotado a un número de personas conocidas, y redes no permisionadas o sin permisos, en estas, cualquiera puede escribir en la Blockchain o crear bloques, para que esta solución sea viable, se suele incentivar de algún modo a los mineros, por ejemplo, ofreciéndoles criptomonedas por cada transacción que procesen.

Para este trabajo usaremos Ethereum, una red privada con permisos usando su tecnología. Para la primera versión, se usará Ganache [7], una red Blockchain personal perteneciente a Ethereum para pruebas y desarrollo muy fácil y cómoda de desplegar. Para una solución profesional, tendríamos que montar nuestra propia red (Esto se desarrollará más en apartados posteriores)

Ilustración 2: Logo de Ethereum

2.2. TRUFFLE-FRAMEWORK

Para el manejo, compilado y despliegue de contratos, se utilizará Truffle [8], un framework muy útil y fácil de usar. Truffle nos permite manejar el ciclo de vida de los contratos, el testeo automático de estos con JavaScript [9], scripts para el despliegue y migraciones y manejar nuestra red de Blockchain entre otras cosas.

Truffle también permite facilidades para el uso de Ganache y React [10], otro framework para la construcción de interfaces de usuario con JavaScript muy utilizado actualmente. Por ello y por otros motivos, construiremos nuestra aplicación con React y NodeJS.

Ilustración 3: Logos de las tecnologías de Truffle-Framework

2.3. DRIZZLE

Drizzle [11] es un conjunto de librerías que nos ayudarán a crear nuestra DApp. Drizzle usa Web3-JS [12], otro conjunto de librerías que nos permite comunicarnos con los nodos de nuestra red de Blockchain. Web3-JS usa un protocolo llamado HTTP-RPC, similar a REST, suele usarse para invocar métodos o funciones de una máquina desde otra. Este protocolo suele ir de la mano de la notación JSON (JavaScript Object Notation) [13], a este protocolo se le llama JSON-RPC [14].

Drizzle también implementa el uso de Redux (React y Flux) [15], una arquitectura para contener el estado de las aplicaciones, acceder a este y modificarlo.

Ilustración 4: Pila de librerías usadas por Drizzle

2.4. SOLIDITY

Solidity [16] es el lenguaje de programación con el que crearemos los contratos, estos contarán con un extensión .sol. Es un lenguaje muy similar a JavaScript en el que los contratos (tipo Contract) son el equivalente a las clases Java (tipo Class), está orientado a contratos. Es bastante reciente, actualmente acaban de sacar la versión v.0.5.9. Por este motivo, aún tiene algunos errores o algunos aspectos que se podrían mejorar. Un claro ejemplo es el tipo de dato Float o Double o número real, el problema con este tipo de dato es que Solidity no lo soporta por el momento.

Ilustración 5: Logo de Solidity

2.5. NODEJS Y NPM

NodeJS [17] es un entorno de JavaScript diseñado para la construcción de aplicaciones Web escalables que junto a npm [18], una herramienta para el manejo de librerias JavaScript de código libre, nos hará posible la creación de nuestra aplicación.

Ilustración 6: Logos de NodeJS y npm

2.6. LIBRERÍAS Y APIS ADICIONALES

En este apartado, definiré y explicaré algunas librerías disponibles en npm y APIs que me han sido de utilidad para el desarrollo de la aplicación:

- Librería QRCode.React [19]: Una librería que te proporciona un componente React, el cual es un elemento "canvas" HTML, usado para mostrar gráficas en las páginas web, que muestra un código QR con el valor que se quiera. En este caso, lo he utilizado para representar las distintas direcciones en las que se desplegarán los contratos usados.

Ilustración 7: Ejemplo de Código QR

- Librería ReactStrap [20]: Una librería que nos proporciona componentes React estilizados con la herramienta Bootstrap [21], una librería para la creación de páginas webs adaptables o "responsive".

Ilustración 8: Logo de Bootstrap

- API de Google Maps [22]: Esta API resulta de gran utilidad para el uso de Google Maps. Se utiliza en esta aplicación a la hora de mostrar la localización de los distintos manipuladores, representada por una latitud y una longitud.

Ilustración 9: Logo Google Maps

2.7. IDES O ENTORNOS DE DESARROLLO

Para desarrollar este trabajo, se han empleado 2 IDEs diferentes, ambos de uso gratuito o de código abierto:

- Sublime Text [23]: IDE sencillo de utilizar, aunque se necesitan muchas dependencias extras para que funcione a la perfección. En la fase final del proyecto, se decidió cambiar de IDE ya que el uso de los colores para destacar el código confundía demasiado.

Ilustración 10: Logo de SublimeText

- Visual Studio Code [24]: IDE cómodo y fácil. La descarga de dependencias es mucho mejor ya que puedes ver la documentación de estas a través del IDE.

Ilustración 11: Logo de Visual Studio Code

3. ANÁLISIS Y DISEÑO

Durante las siguientes secciones recurriré en varias ocasiones a la notación UML [25], la cual hace uso de varios elementos para describir el sistema a analizar. Por ejemplo, los diferentes actores que interactúan con el sistema, los diferentes casos de uso y sus relaciones, etc.

3.1. REQUISITOS DEL SISTEMA

En este apartado se describirán los diferentes requisitos de calidad o restricciones que deberá cumplir el sistema, tanto la primera versión desarrollada como la futura versión profesional. Se seguirá el estándar ECSS-E-ST-40C [26] para tratar los siguientes requisitos no funcionales:

- **Prestaciones:** Este aspecto no ha importado mucho a la hora de desarrollar la primera versión, pero para una versión profesional será importante. Este requisito involucra capacidades, tiempos de carga, frecuencia, velocidad, etc.
- **Interfaces:** La interfaz con la que interactúa esta aplicación está clara: La red de Blockchain de Ethereum. En la primera versión será Ganache (red de pruebas y desarrollo) pero en la profesional sería una red privada propia o máquinas alquiladas a través de AWS.
- **Operación:** En la primera versión, todos los usuarios accederán al sistema a través de una aplicación basada en interfaces Web. Sin embargo, esto dista mucho de una posible versión profesional. Lo ideal sería que los clientes accedieran a través de aplicaciones de móvil que hagan uso de las cámaras para capturar los códigos QR asociados a cada producto, tras el escaneo de este código se mostraría la información del producto. El administrador podría seguir accediendo a través de la aplicación Web sin ningún problema. Los manipuladores, por otra parte, necesitarán de dispositivos especiales, por ejemplo, escáneres de códigos QR con pantalla y teclado, el manipulador al escanear el código QR del producto con el que está interactuando, el dispositivo se encargaría de hacer las interacciones con la red de Blockchain. Este tendría programado la dirección del contrato inteligente referente al manipulador para facilidad del manipulador. El manipulador mediante la pantalla y teclado indicaría los datos a guardar en la Blockchain.
- **Recursos:** Como con las prestaciones, en la primera versión no se han tenido en cuenta los recursos, pero en la versión profesional será un requisito crítico ya que los recursos de computación, almacenamiento, memoria… serán de vital importancia para la viabilidad del sistema. Este apartado se desarrollará en mayor profundidad en el apartado referente a "Líneas futuras".
- **Verificación:** Como se verá más adelante, se han realizado algunas pruebas para verificar el correcto funcionamiento de la primera versión. El sistema final requerirá de pruebas más exhaustivas ya que la complejidad del sistema aumentará.
- **Pruebas de aceptación:** Estas afectarán a la puesta en operación del sistema. Por esto, en la primera versión no se han tenido en cuenta, aunque harán falta en el despliegue profesional.
- **Documentación:** Esta memoria actuará como documentación, a parte de la bibliografía al final descrita.
- **Seguridad:** Este aspecto es uno de los más importantes al interactuar con tecnologías y soluciones basadas en Blockchain. El uso de Blockchain ya garantiza la integridad de la información guardada en ella, así como su autenticidad tanto en la primera versión como en la profesional (Esto se debe a los "hashes"). En esta primera versión desarrollada, sin embargo, la disponibilidad y autorización no se tienen en cuenta. Todo el sistema se ejecuta y opera en

una única máquina, si esta falla, todo el sistema falla. No existe un sistema de "LogIn" para administradores o manipuladores así que todo el mundo está autorizada al uso del sistema actuando como cualquier tipo de usuario, desde administrador hasta cliente. Esto no puede pasar en un despliegue profesional, serán necesarias varias máquinas o servidores para albergar la aplicación y nodos de la red de Blockchain y un sistema de autentificación pertinente. Por ejemplo, el uso de la aplicación móvil te autoriza al uso del sistema como cliente, la posesión y el uso del dispositivo especial que tendrán los manipuladores les autoriza a usar el sistema, y, por último, un sistema de "LogIn" a través de un portal Web podría ser una solución para los administradores.

Cabe destacar un par de aspectos. Primero, la creación y escritura de Smart Contracts está restringido solo a administradores y manipuladores, esto se consigue a través de Solidity. Programando un campo en cada contrato de tipo "address" llamado "owner" (se explicará en mayor profundidad en secciones posteriores) el cual se le asigna la dirección de su creador en el momento de la construcción de un contrato. Aunque en la primera versión, este campo no se usa para aportar seguridad, será necesario modificar los contratos para un despliegue profesional. Simplemente habría que añadir una pequeña comprobación en cada método para ver que la persona que quiere modificar el contrato o eliminarlo es aquel que lo ha creado, un administrador o un manipulador. Segundo, la autenticidad e integridad de la información recaerá en el grado de confianza que se tenga con los manipuladores, ya que la información que ellos publiquen no se podrá modificar si ellos no lo desean (o en su defecto, el administrador del sistema). Si estos manipuladores extravían su escáner o publican información falsa es algo con lo que Blockchain no puede lidiar (en caso de extravío del dispositivo, se podrían tomar medidas como inhabilitar la dirección programada en ese dispositivo, así este quedaría inutilizado y no podría causar mal alguno).

- **Portabilidad:** En la primera versión, la aplicación web deberá funcionar al menos en los navegadores más usados hoy en día (Chrome, Mozilla Firefox, Microsoft Edge y Safari). En la versión final, se crearán aplicaciones que funciones tanto en Android como en IOs.
- **Fiabilidad:** En la primera versión no se ha tenido en cuenta este aspecto, aunque se ha desarrollado evitando cualquier fallo posible, en la versión profesional del sistema será importante.
- **Mantenibilidad:** En la primera versión, al no estar previsto desplegarla, este requisito no nos afecta. Por otro lado, en la versión final, este requisito será importante ya que cualquier aplicación actual tiene que adaptarse a la perfección a los cambios tras el despliegue. Hay que destacar que el mantenimiento y operación de las aplicaciones o servicios en uso requiere la misma cantidad de esfuerzo que el desarrollo sino más.
- **Salvaguarda o safety:** El uso de este sistema no debería producir daños personales o materiales. El único daño que pudiese producir sería a través de la escritura de datos "sensibles" erróneos sobre productos, como puede ser la fecha de caducidad. Si un manipulador establece una fecha equivocada, ya sea por error o con maldad, esto podría llegar a afectar al que consumiese el producto supuestamente en buen estado cuando no lo esté. Un error humano en muchas ocasiones no se puede evitar, pero las malas intenciones dependerán una vez más de la confianza que depositemos en el manipulador.

3.2. ARQUITECTURA DEL SISTEMA

Este apartado mostrará las diferentes arquitecturas que este trabajo tratará:

- Una primera versión, desarrollada completamente por mí.
- Una versión por desarrollar en un entorno profesional si se diera el caso.

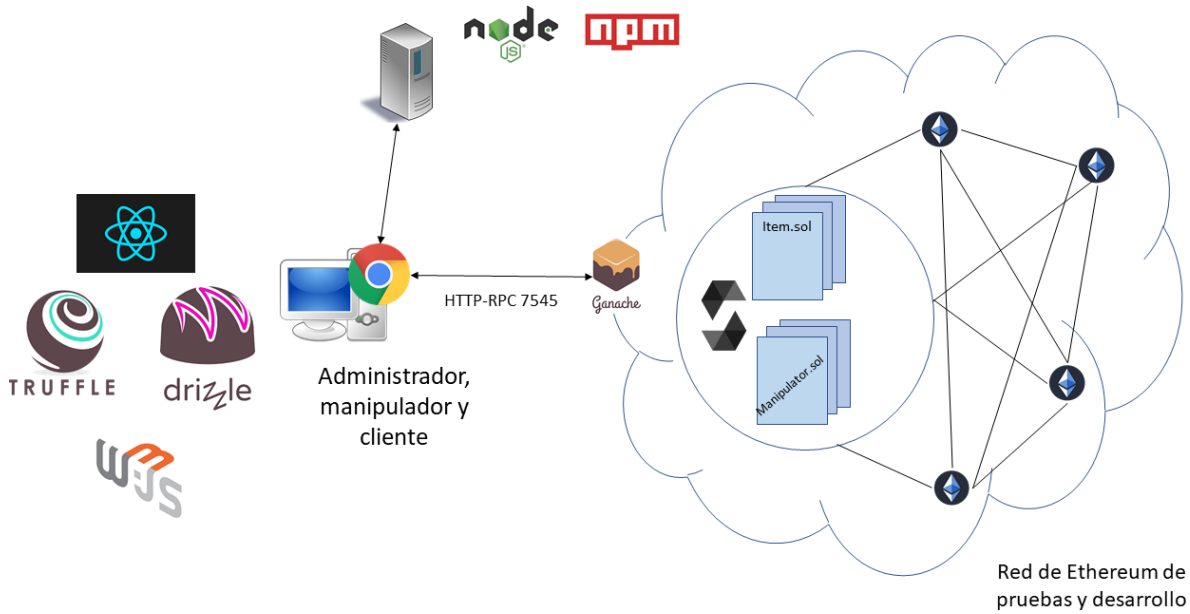

Ilustración 12: Arquitectura y tecnologías usadas en la primera versión desarrollada

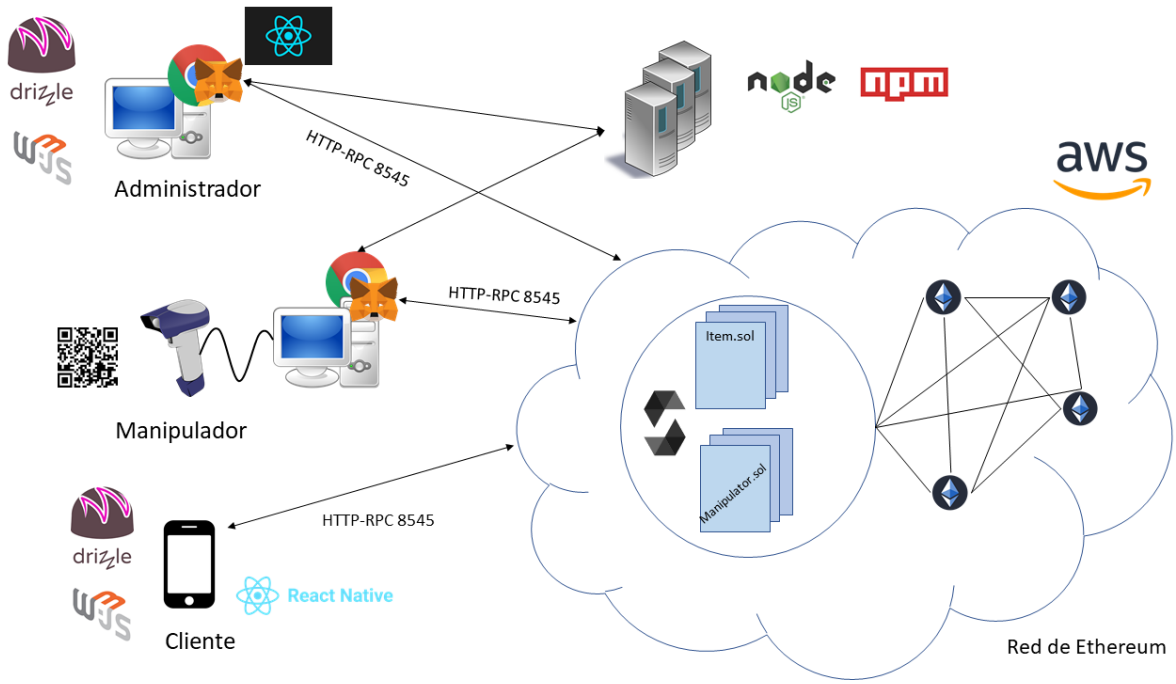

Ilustración 13: Posible arquitectura de un despliegue profesional

Nota: La tecnología que usa el manipulador y administrador sobre Chrome se llama Metamask [27]*, es una extensión de Chrome que te permite comunicarte con las redes de Ethereum.*

Podemos ver claras diferencias entre ambas arquitecturas, pero la esencia es la misma, ambas usan las mismas tecnologías o alternativas muy similares (es el caso de React-Native [28] y las redes de Ethereum, en la primera versión se usa Ganache y en la versión profesional se usaría una red privada para despliegues). Al usar Ganache, el puerto de acceso a esta red (:7454) cambia respecto de una red privada de Ethereum (:8545).

3.3. DESCRIPCIÓN DEL SISTEMA

A continuación, se puede ver el modelo de dominio del sistema representado con un diagrama de clases UML:

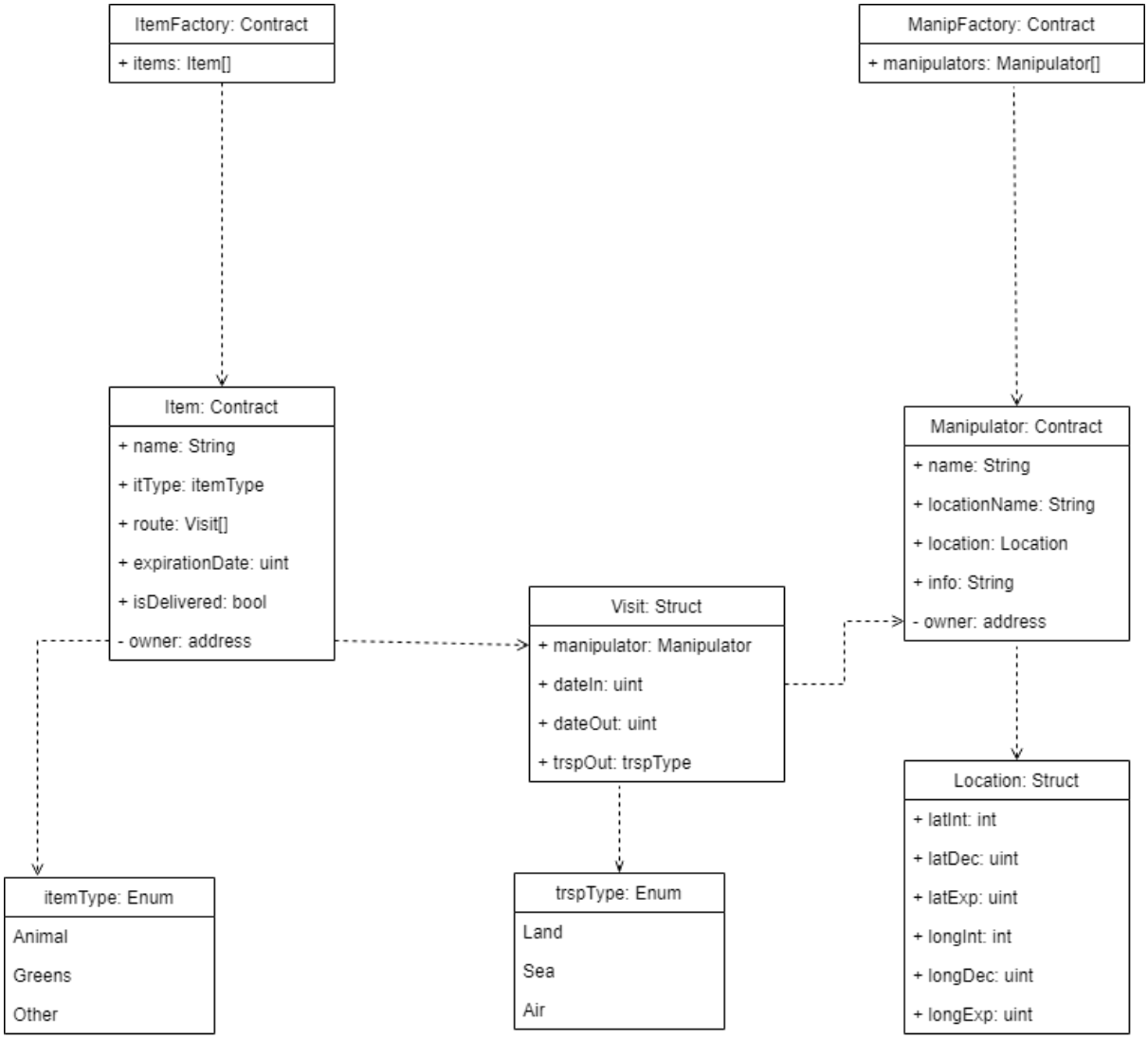

Ilustración 14: Dominio del Sistema desarrollado y estructura de datos usada en los contratos inteligentes

Partimos de dos contratos padre escritos en Solidity (equivalentes a clases en Java). Estos serían las dos factorías: ItemFactory y ManipFactory. Se encargarán de almacenar y administrar los distintos productos y manipuladores respectivamente de nuestro sistema. Luego, tenemos Item y Manipulator, contratos que actuarán de modelos de datos. Con sus distintos atributos almacenaremos los datos que se manejarán y serán visibles para el cliente. El único atributo privado que poseen es owner, que representa la dirección del contrato que ha creado el Item o el Manipulator. Sirve para temas de permisos y roles: solo el contrato que ha creado al Item o el Manipulador puede eliminarlos y cuando estos son eliminados, los fondos almacenados en el contrato irán a parar a owner (ItemFactory o ManipFactory).

Por otra parte, haré uso de varios enumerators (itemType y trspType), que representan el tipo del alimento y el tipo de transporte por el que viajarán. También usaré dos Structs, uno en cada contrato Item y Manipulator. Visit representará los distintos hitos en la ruta que seguirá un producto y Location la posición exacta del manipulador (Latitud y Longitud). Debido a que solidity no maneja Doubles o números reales, he tenido que arreglármelas para conseguir un tipo "Double" propio. Por ejemplo, obtendríamos la latitud de la siguiente forma:

$$Latitud = latInt \pm latDec * 10^{-latExp}$$

La suma o resta de la parte decimal dependerá del signo de latInt, si es mayor que 0, se sumará la parte decimal y se restará en caso contrario. La longitud se obtendría de una forma análoga a la anterior.

3.4. CASOS DE USO

Comenzaré describiendo los diferentes casos de uso, los cuales definen las diferentes acciones que se pueden realizar en el sistema. Se muestran en el siguiente diagrama:

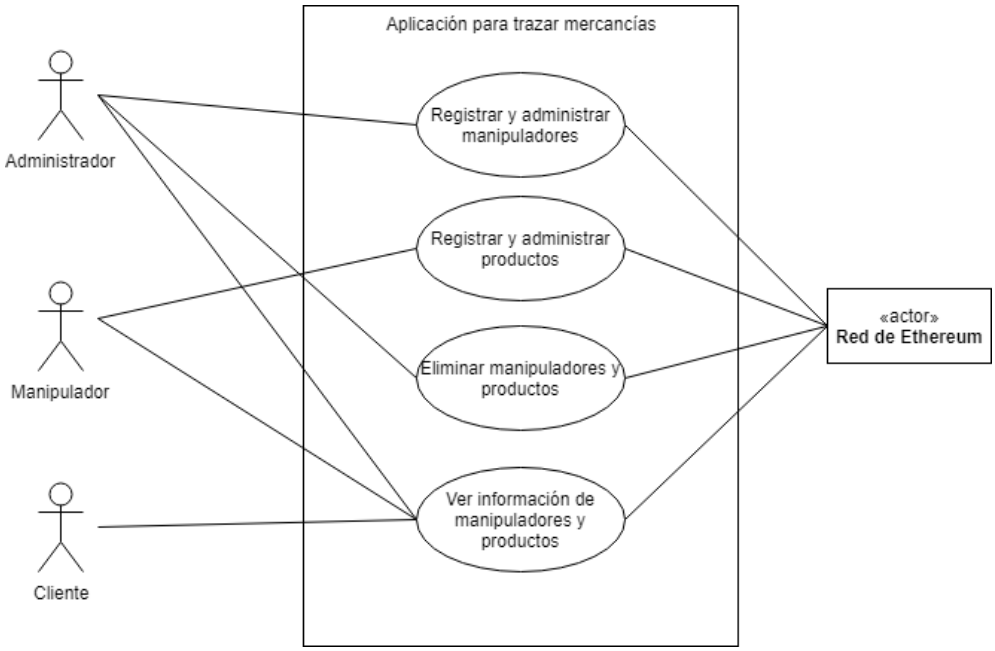

Ilustración 15: Diagrama de casos de uso

3.4.1. CASO DE USO PRINCIPAL

El caso de uso más importante y que aporta más valor a la aplicación sería el de ver la información almacenada en el sistema acerca de los productos y manipuladores registrados.

Actores: Administrador, manipulador y cliente. Todos los actores del sistema pueden acceder a la información en cualquier momento.

Actor secundario: Red de Ethereum con la que se interactúa continuamente.

Descripción: Este caso de uso trata las diferentes consultas que realizan los actores al sistema. Sin embargo, hay una pequeña diferencia entre los distintos actores. Tanto administradores como manipuladores pueden consultar cualquier información de un producto independientemente del estado del producto. Los clientes solo pueden visualizar los productos si este ha llegado a su último hito en la ruta de transporte, es decir, un cliente solo puede ver los productos que tiene disponible para comprar en su supermercado. (El atributo del contrato Item responsable de representar esto, si un producto está o no visible para el cliente, es isDelivered. Si es TRUE será visible y si es FALSE no será visible)

3.4.2. CASOS DE USO SECUNDARIOS

El resto de los casos se clasificarían como secundarios y serían los siguiente:

Caso de uso 1: Registrar y administrar manipuladores.

Actores: Administrador. El único con permisos para llevar a cabo este caso de uso es el administrador del sistema.

Actor secundario: Red de Ethereum con la que se interactúa continuamente.

Descripción: Este caso de uso representa las diferentes acciones que tiene que tomar un administrador para registrar un manipulador en la red de Blockchain para que este tenga permiso para registrar productos y modificarlos.

Caso de uso 2: Registrar y administrar productos.

Actores: Manipuladores. Los únicos capaces de realizar este caso son los diferentes manipuladores registrados en el sistema (Productores, distribuidores…).

Actor secundario: Red de Ethereum con la que se interactúa continuamente.

Descripción: Este caso de uso representa las diferentes acciones que tienen que tomar los diferentes manipuladores para registrar un producto en la red de Blockchain. Cuando este producto llegue al supermercado y se actualice su información, quedará disponible para consultas por parte de los clientes.

Caso de uso 3: Eliminar manipuladores y productos.

Actores: Administrador. El único con permisos para llevar a cabo este caso de uso es el administrador del sistema.

Actor secundario: Red de Ethereum con la que se interactúa continuamente.

Descripción: Este caso de uso representa las diferentes acciones que tiene que tomar un administrador para eliminar un manipulador o producto de la red de Blockchain, ya sea porque el manipulador ya no tiene derechos o porque un producto haya caducado.

3.5. DIAGRAMA DE SECUENCIA DEL SISTEMA

Para representar la interacción de los distintos actores con el sistema he elegido un diagrama de secuencia. Es el siguiente:

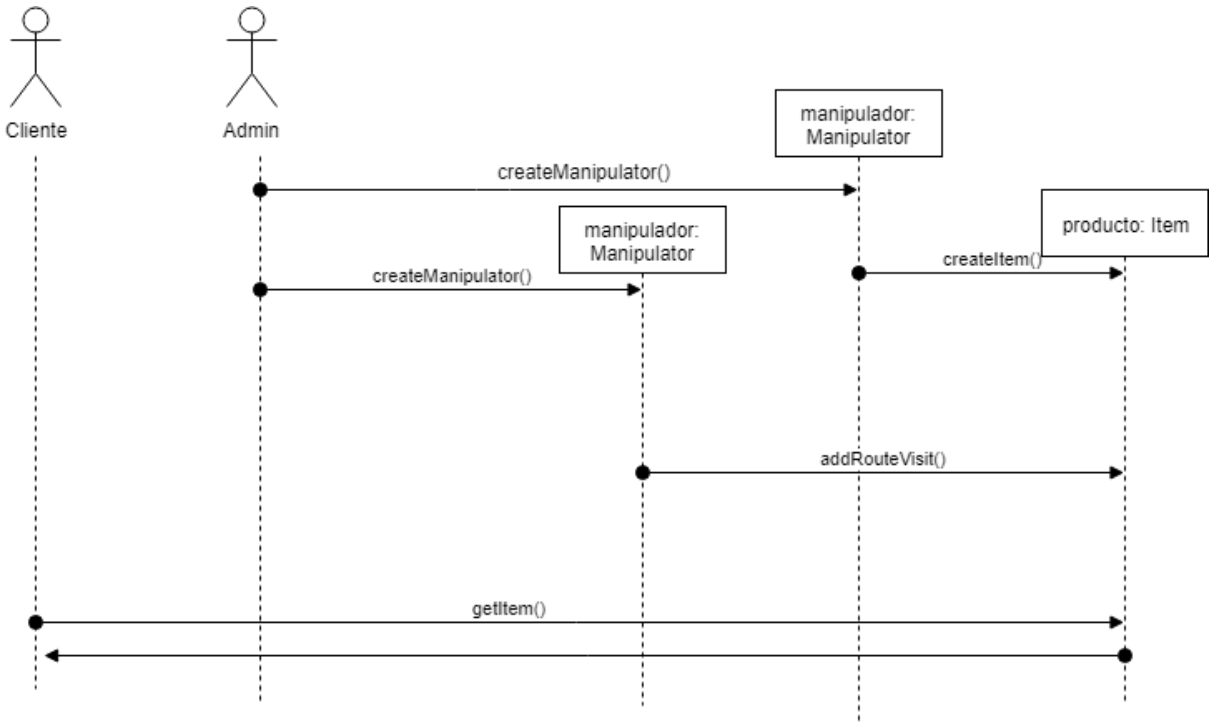

Ilustración 16: Diagrama de secuencia de las acciones tomadas por los distintos actores

Partimos de una ejecución desde cero, es decir, es la primera vez que se ejecuta la aplicación y no hay nada creado.

En primer lugar, el administrador del sistema tiene que crear, mediante el contrato de ManipFactory, los distintos contratos Manipulator para facilitar el acceso de los manipuladores al sistema. Una vez creados los manipuladores, estos pueden crear contratos Item y modificarlos. Por ejemplo, uno registra su producto en el sistema creando un contrato para este y otro manipulador, el distribuidor por decir uno, añade un hito a la ruta del Item.

Una vez acabada la interacción de los distintos manipuladores con el producto, este llegaría al supermercado y estaría disponible para los clientes. Estos, podrían usar una aplicación de móvil o una aplicación web para comprobar la información de origen y ruta del producto.

Con esto, concluiría el análisis de nuestro sistema y aplicación. En otras ocasiones, con aplicaciones más complejas, sería adecuado mostrar un diagrama de estados representando la navegación a través de la web. En mi caso, es una “Single Page Application” con una navegación muy simple: Un cliente abre su aplicación, comprueba la información del producto y cierra la aplicación.

4. IMPLEMENTACIÓN Y DESARROLLO

En este apartado, me centraré en describir los distintos pasos que he seguido para desarrollar esta aplicación. El desarrollo se podría dividir en dos partes: La primera, en la que desarrollo contratos y aplicaciones sencillas para tener un primer contacto con todas las tecnologías a usar (Se siguen los pasos descritos en el Anexo D), y la segunda, en la que ya implemento y uso todos los conocimientos adquiridos en la fase previa, y desarrollo la que será la primera versión de la aplicación final.

4.1. IMPLEMENTACIÓN

El siguiente desarrollo está desplegado en mi perfil de Github:

https://github.com/Jaimedfc/TFG

Empecé pensando qué información podría interesar a los futuros clientes de la aplicación. Para ello, vi varios ejemplos que han desarrollado las grandes marcas como Carrefour [29]. Al final, me decanté por la información que actualmente se puede ver en la versión final:

- Para los manipuladores:
  - Nombre.
  - Lugar (País, zona, municipio…).
  - Localización (Latitud y Longitud).
  - Información extra (acciones que realizan, cómo interactúan con los productos…).
- Para los productos:
  - Nombre.
  - Tipo de producto (Animal, cultivo u otro).
  - Fecha de caducidad.
  - Si ha llegado o no al destino final.
  - Ruta que ha seguido. Cada hito en la ruta se representa con distintos datos como el manipulador que entra en contacto con el producto durante ese hito, fecha de llegada al manipulador, fecha en la que el manipulador distribuye la mercancía y el tipo de transporte con el que la distribuye (Tierra, mar o aire).

Una vez conseguido un modelo de datos aceptable, comencé con la implementación.

En primer lugar, programé los contratos Manipulator y ManipFactory. Los relativos a los productos ya estaban prácticamente hechos (Tras haber realizado los pasos del Anexo D), lo único que les faltaba eran los campos del modelo de datos final y una forma de eliminar contratos. La eliminación de contratos era algo que no había tenido en cuenta hasta que lo mencionó mi tutor. Es algo fundamental en una aplicación como esta, ya que, si nos dedicamos a crear contratos infinitos, sin borrar ninguno, en algún momento la eficiencia de la aplicación se podría ver afectada negativamente. Desarrollé una forma de eliminar contratos, aunque de una forma bastante ineficiente. Simplemente cuando quería eliminar un Item o un Manipulador, lo buscaba en el array de su factoría correspondiente y sustituía su valor en el array por un Item o Manipulador desplegado en una dirección inexistente o cero(0x0). En el apartado Front-End, eliminar un contrato implica que Drizzle tiene que dejar de observar un contrato. Esto se consigue añadiendo el código indicado en el apartado “Removing Contracts Dynamically” de la siguiente referencia [30] en el método al que llamamos al pulsar el botón de “Eliminar Elemento”.

Por esto, en el apartado Back-End o Solidity, a la hora de crear contratos, primero busco en el array a ver si encuentro alguna dirección cero y la sobrescribo con el contrato creado, si no encuentro ninguna dirección a cero, añado el nuevo elemento al final del array.

Tras acabar con el desarrollo en Solidity, me enfoqué en la parte de JavaScript o de presentación. Desarrollé varias vistas para la aplicación, una para los distintos actores que interactuarán con el sistema: Administradores, manipuladores y clientes.

Durante este desarrollo final, encontré varias complicaciones que me llevaron más tiempo del pensado. Lo más destacable fue el manejo de fechas, en un primer lugar, había pensado guardar en la Blockchain un único número que representase la fecha. En JavaScript existe un método aplicable al tipo Date el cual es getTime(), este método devuelve el número de milisegundos que han transcurrido desde el 1 de Enero de 1970 hasta la fecha sobre la que se aplica el método. Esto me resulta de gran utilidad ya que el constructor de Date acepta milisegundos para su construcción, es decir, si sabes el número de milisegundos desde el 1 de Enero de 1970 que han transcurrido hasta la fecha que quieres obtener simplemente puedes hacer: fecha = new Date(milisegundos). Hasta aquí, ningún problema, guardaba este valor en la cadena y lo obtenía, sin embargo, a la hora de crear el objeto Date una vez recuperado los milisegundos, me encontraba con un error. Tras varias pruebas, me di cuenta de que escribía un número erróneo en la cadena y por ello al recuperarlo y crear la fecha, obtenía una fecha invalida.

En último lugar, lo único que falta por hacer es aplicar elementos estéticos a la aplicación, es decir, usar un CSS atractivo para que llame la atención. Para este apartado, decidí usar Bootstrap, una herramienta muy versátil a la hora de crear páginas web adaptables o "responsive" y estéticas. También introduje elementos típicos de una web, como pueden ser la barra de navegación, logos y un pie de página o "footer" para darle un aire de profesionalidad a la página y que no se base simplemente en formularios y texto plano. Además, introduje imágenes para diferenciar el tipo de los productos y otra imagen para representar a los manipuladores.

Una vez acabada esta parte, decidí implementar alguna función interesante ya que tenía el tiempo necesario y me apetecía hacer más atractiva la aplicación. La primera función fue la de sustituir todas las direcciones que se mostraban por su representación en código QR. Gracias a una librería llamada "QRcode.react", se pudo realizar con facilidad. La segunda función que implementé fue el uso de la API de Google Maps para mostrar en un mapa las coordenadas almacenadas en los contratos. Todas estas tecnologías ya están descritas en el apartado de "Estado del Arte".

Con estas últimas pinceladas puedo dar por acabada la primera versión de mi aplicación. El resultado de cada tipo de vista lo muestro con capturas de pantalla a continuación:

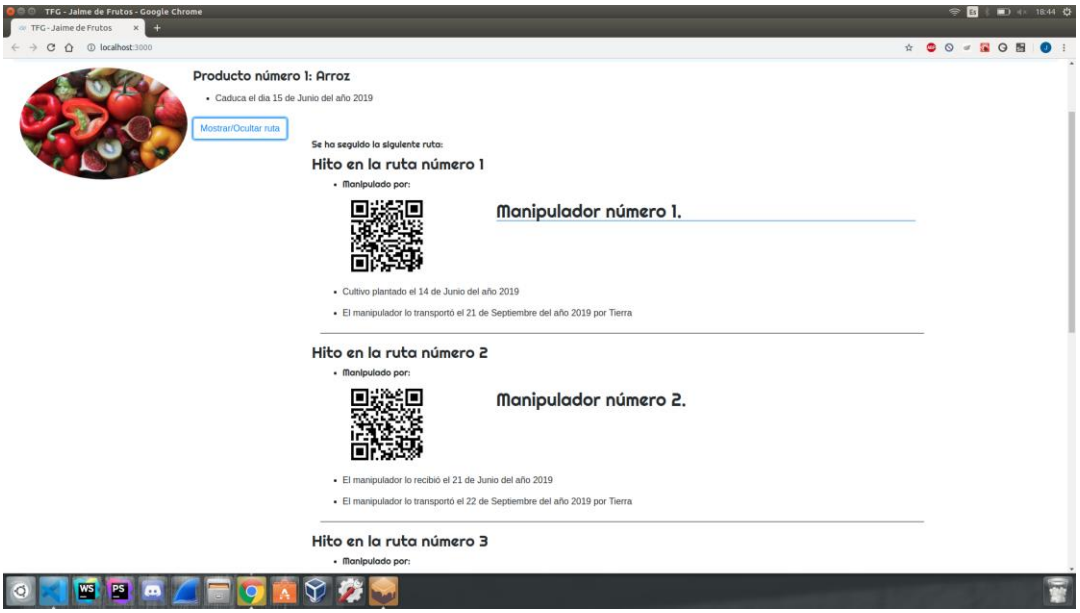

Ilustración 17: Vista de cliente de un producto

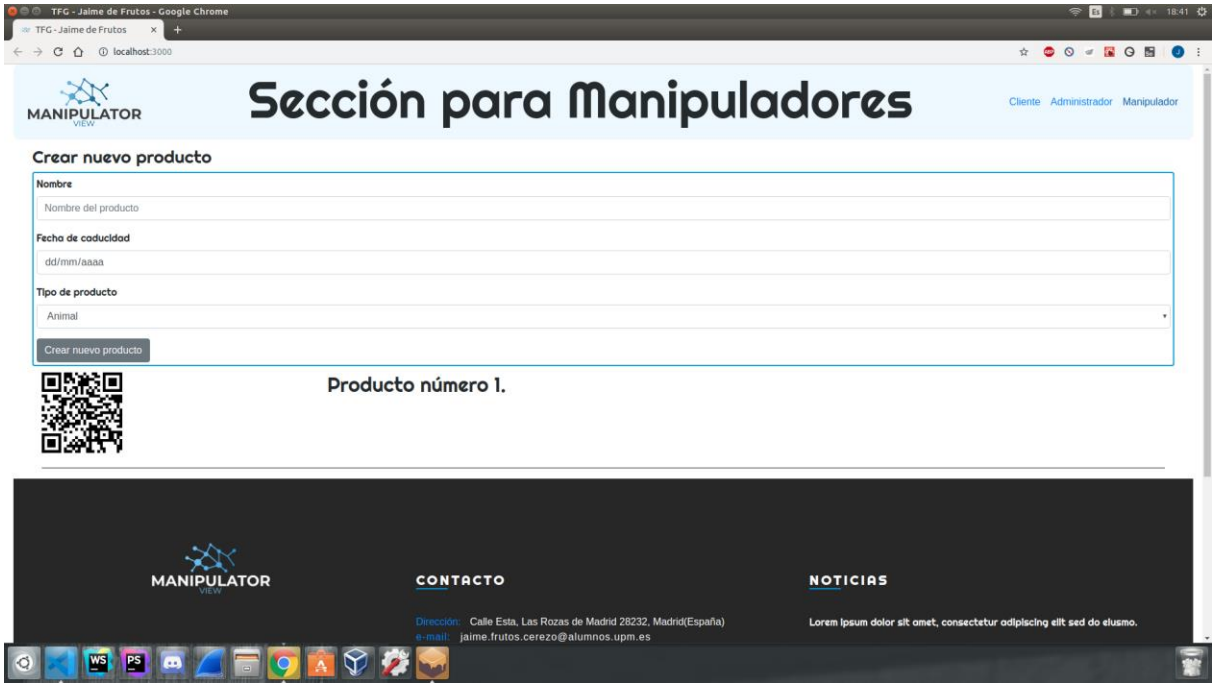

Ilustración 18: Vista de manipulador con formulario para registrar productos

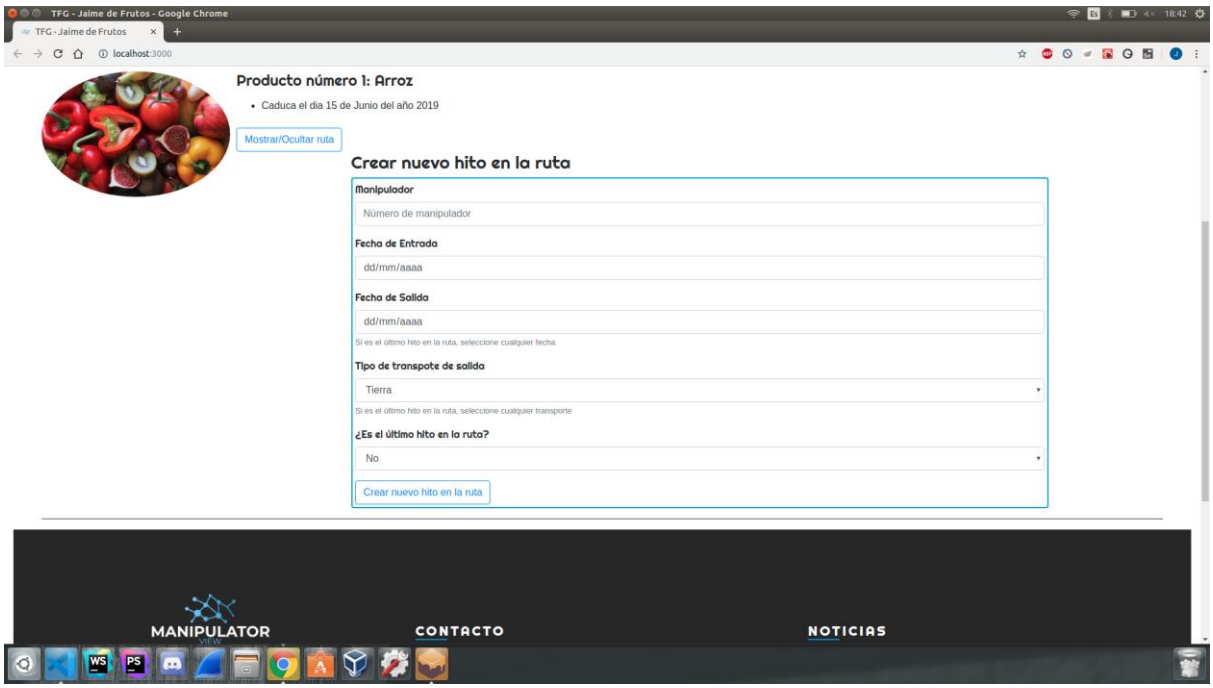

Ilustración 19: Vista de manipulador con formulario para registrar hito de ruta de un producto

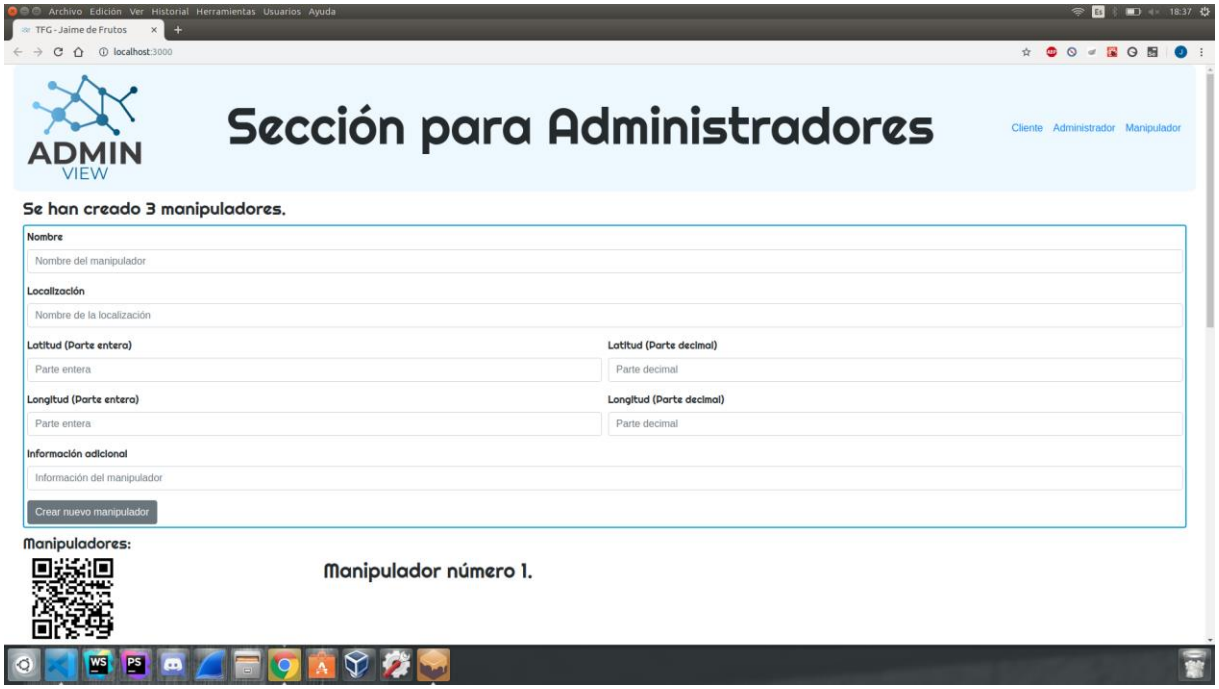

Ilustración 20: Vista de administrador con formulario para registrar manipuladores

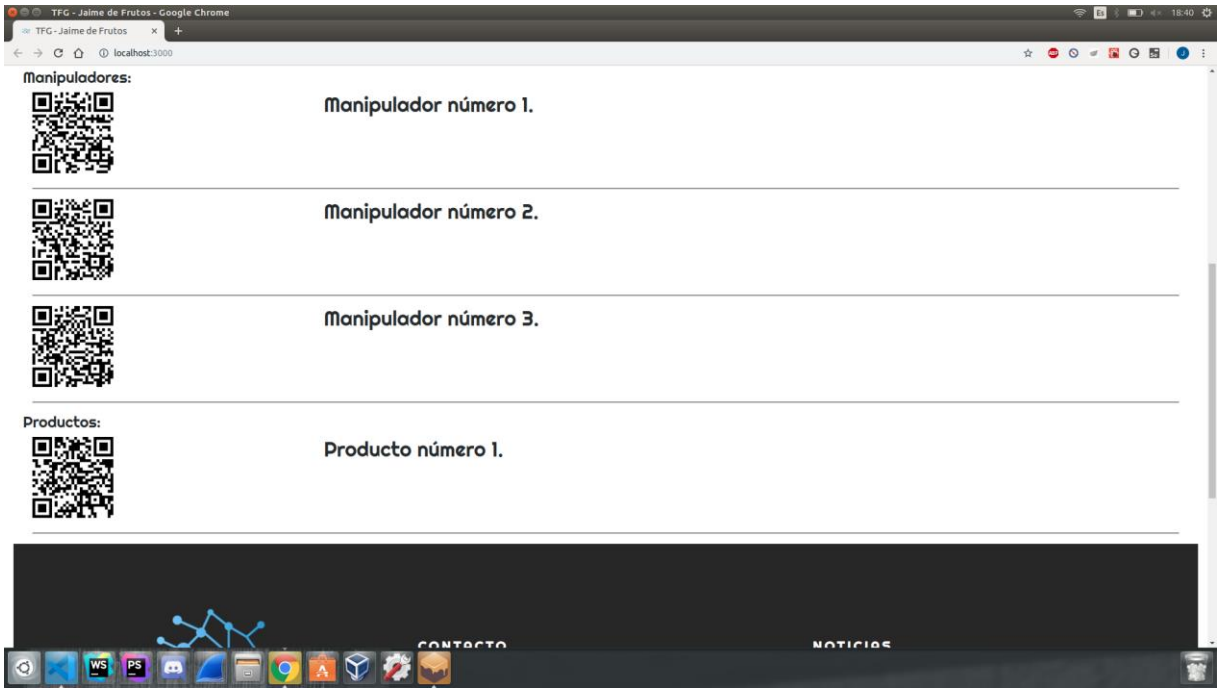

Ilustración 21: Vista de administrador con dos listas con todos los elementos registrados en la red de Blockchain

Las siguientes versiones podrían implementar más funciones a parte de las ya descritas. Estas líneas futuras las dejaré para una sección posterior.

4.2. DESCRIPCIÓN DEL PROYECTO

Para finalizar con la implementación, se procederá a explicar la función de cada fichero y directorio que podremos encontrar en el proyecto subido a GitHub. La estructura del proyecto es la siguiente:

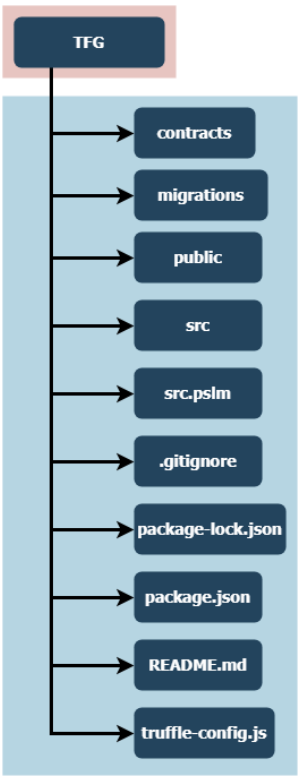

Ilustración 22: Estructura del Proyecto subido a GitHub

Al clonar el proyecto, nos encontramos con el directorio que lo recoge, TFG. Dentro de este nos encontramos con:

- **Directorio contracts:** Directorio creada por el framework Truffle con el comando:

```
truffle init
```

  desde consola. En él, se encuentra todos los contratos usados en la aplicación escritos en Solidity.

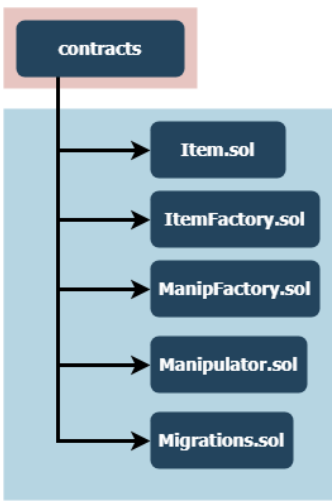

Ilustración 23: Estructura del directorio contracts/

- **Directorio migrations:** Directorio creado al igual que contracts. Contiene las migraciones usadas en el despliegue de los contratos en la Blockchain. Contiene dos migraciones:
    - **1_initial_migration.js:** Migración autogenerada para el despliegue de Migrations.sol, contrato generado por Truffle. Tanto este contrato, como esta migración, son necesarios para poder migrar nuestros propios contratos.
    - **154953303_deploy_factory.js:** Migración creada a través del comando:

```
truffle create --migration deploy_factory
```

Con esta migración, desplegamos en la Blockchain nuestros dos contratos factoría, ItemFactory.sol y ManipFactory.sol.

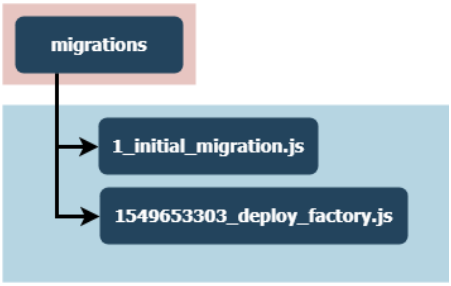

Ilustración 24: Estructura del directorio migrations/

- **Directorio public:** Este directorio es creado mediante otro comando desde consola:

```
create-react-app TFG
```

Este comando crea un directorio llamado TFG con un esqueleto dentro de archivos y directorios predeterminado para comenzar a desarrollar una aplicación con React. Una vez dentro del directorio TFG ejecutaríamos el comando descrito en el directorio contracts para hacer que esta carpeta sea manejable por el framework de Truffle. La carpeta public contiene todo el material que va a ser visible por los clientes: imágenes, iconos, página principal index.html…

El archivo manifest.json sirve para configurar varios aspectos como ruta inicial, nombre del archivo que se tomara como icono, tamaños…

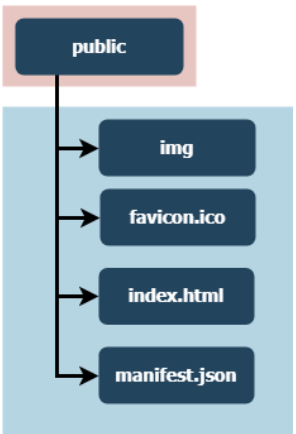

Ilustración 25: Estructura del directorio public/

- **Directorio src y src.pslm:** El directorio src sustituye a la carpeta src.pslm, la cual es creada por el comando descrito en el directorio anterior. src.pslm contiene bastantes utilidades que no se utilizan en esta aplicación pero que no se han borrado para un posible futuro. src contiene los componentes desarrollados de React, ficheros CSS, el archivo drizzle.js para configurar la "store" y el contexto de Drizzle para gestionar los contratos desplegados y el estado, el archivo index.js con el cual definimos un "Provider" que nos proporciona el objeto drizzleContext, desde el cual, en el componente App.js, obtendremos gracias a un "Consumer" los objetos drizzle, drizzleState e initialized que usaremos en la aplicación con bastante frecuencia. Al final de este apartado, se describirá la estructura de los componentes, así como su funcionalidad.

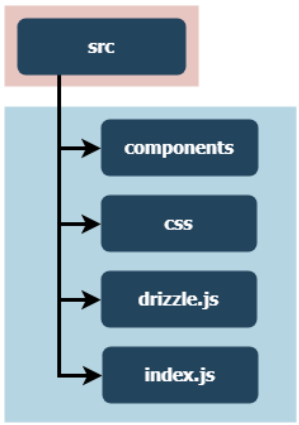

Ilustración 26: Estructura del directorio src/

- **Fichero .gitignore:** Fichero en el cual describir qué archivos o directorios se deben ignorar por la herramienta de control de versiones git [30].
- **Fichero package-lock.json:** Fichero autogenerado por NodeJS para manejar las versiones de las distintas dependencias utilizadas.
- **Fichero package.json:** Fichero autogenerado por NodeJS en el cual indicar las dependencias o librerías usadas, scripts… Este archivo es de vital importancia, nos evita tener que manejar las librerías con git, las cuales se almacenan en un directorio llamado node_modules, el cual se crea al ejecutar el comando:

```
npm install
```

- **Fichero README.md:** Fichero generado por GitHub al crear el repositorio. En él, podemos escribir información referente al proyecto para informar al que quiera utilizarlo en un futuro. Por ejemplo, podemos escribir una especie de "Manual de uso" para la instalación y ejecución del proyecto entre otras cosas.
- **Fichero truffle-config:** Fichero utilizado y generado por Truffle. En él, establecemos dónde encontrar la red de Blockchain a utilizar o en que carpeta desplegar los contratos compilados.

- **Componentes React:** El árbol de componentes padre e hijos se muestra a continuación:

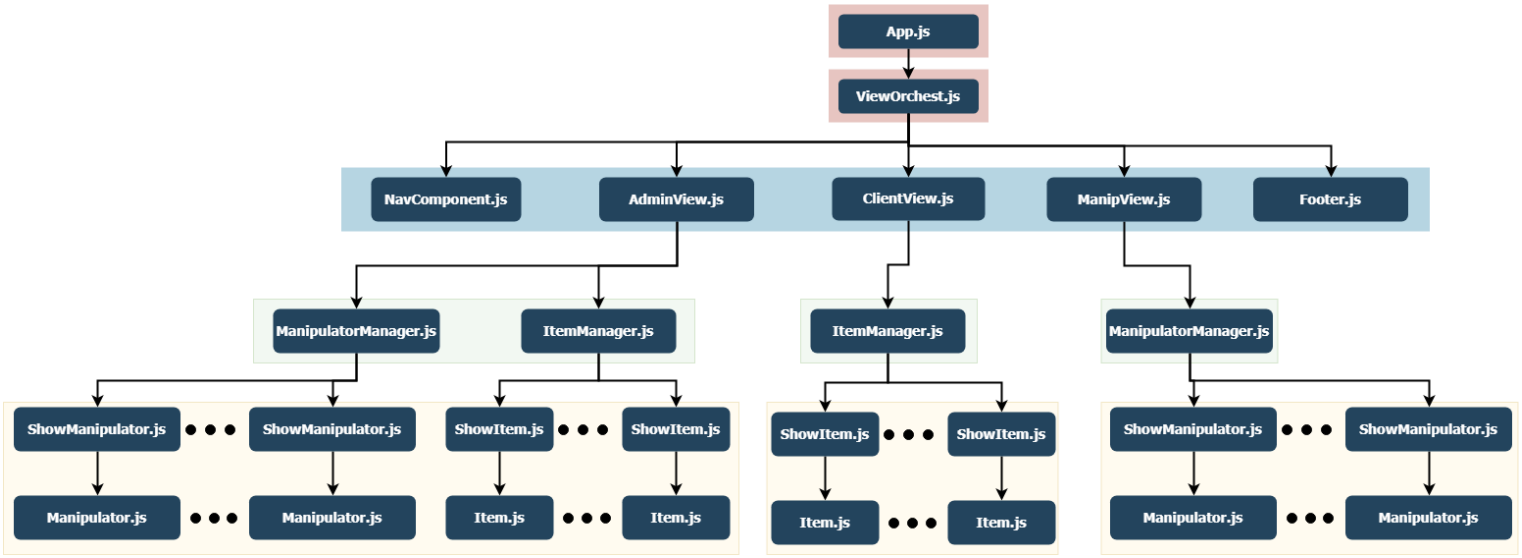

Ilustración 28: Primera parte del árbol de componentes React usados en la aplicación Web

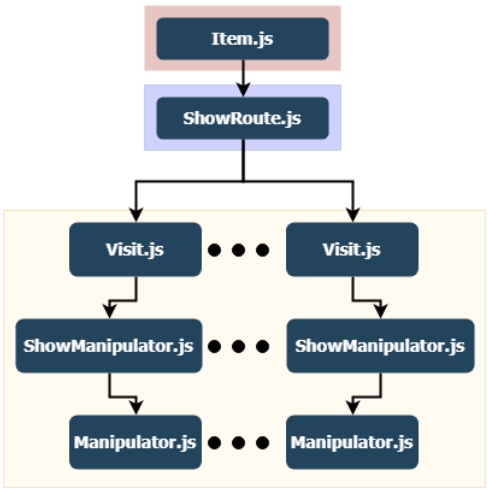

Ilustración 27: Segunda parte del árbol de components React usados en la aplicación Web

En primer lugar, destacaré la estructura de métodos típica de un componente que interactúa con la red de Blockchain:

- Definición de la clase extendiendo a “React.Component”:

```
4   class ClientView extends React.Component
```

Ilustración 29: Definición de la clase del componente ClientView.js

- Estado inicial usado en el componente:

```
7       state = { manipulatorCounterKey:null,
8                 itemCounterKey:null};
```

Ilustración 30: Definición del estado inicial del componente ClientView.js

En este caso, nos encontramos con las claves que recuperaremos más tarde con Drizzle para obtener el número de productos y manipuladores registrados en la Blockchain.

- Método componentDidMount(), el cual se ejecuta al montar el componente y es donde recuperaremos las claves anteriores:

```
componentDidMount() {
const { drizzle } = this.props;
const ManipFactory = drizzle.contracts.ManipFactory;
const ItemFactory = drizzle.contracts.ItemFactory;

var manipulatorCounterKey;
var itemCounterKey;

manipulatorCounterKey = ManipFactory.methods["getManipLength"].cacheCall();
itemCounterKey = ItemFactory.methods["getItemsLength"].cacheCall();


this.setState({ manipulatorCounterKey, itemCounterKey });
console.log(drizzle);
console.log(this.props.drizzleState);

}

```

Ilustración 31: Método componentDidMount() del componente ClientView.js

Recuperamos del objeto drizzle nuestros contratos factoría a usar. Llamamos a los métodos definidos en los contratos para recuperar las claves y las guardamos en el estado para más tarde recuperar los valores guardados con ellas.

- Método render llamado para pintar el componente. En él, usamos las claves para recuperar los valores buscados a través del objeto que nos facilita Drizzle llamado drizzleState:

```
render() {

  const { drizzleState } = this.props;
  const ItemFactory = drizzleState.contracts.ItemFactory;
  const ManipFactory = drizzleState.contracts.ManipFactory;

  var manipulatorsLength = ManipFactory.getManipLength[this.state.manipulatorCounterKey];
  var itemsLength = ItemFactory.getItemsLength[this.state.itemCounterKey];


   return (
     <div>

       <ItemManager drizzle={this.props.drizzle}
         drizzleState={this.props.drizzleState}
         isManipulator={false}
         itemsLength={(itemsLength && itemsLength.value) || 0}
         manipLength={(manipulatorsLength && manipulatorsLength.value) || 0}
       />
     </div>
   );
}
```

Ilustración 32: Método render() del componente ClientView.js

Para más información acerca del ciclo de llamadas de los métodos de un componente React, adjunto la siguiente referencia [32].

Una vez conocemos la estructura media de los componentes más complejos, procederé a describir lo que hace cada componente brevemente:

- **App.js:** Como se ha comentado anteriormente, en App.js configuramos un "Consumer" para obtener los objetos que nos proporciona Drizzle.

- **ViewOrchest.js:** Primer componente de nuestra aplicación, con el controlamos qué vista mostrar, además de mostrar una barra de navegación y un pie de página.

- **NavComponent:** Barra de navegación de nuestra aplicación con los botones necesarios para movernos entre vistas.

- **Footer.js:** Pie de página de nuestra aplicación.

- **AdminView.js:** Vista a mostrar a los administradores. Nos muestra un formulario para crear manipuladores (Smart Contracts), y dos listas: Una para ver y manejar los manipuladores registrados en la Blockchain y otra para ver y manejar los productos también registrados.

- **ClientView.js:** Vista a mostrar a los clientes, la cual nos muestra todos los productos de la Blockchain.

- **ManipView.js:** Vista a mostrar a los manipuladores que usen la aplicación. Muestra un formulario para registrar productos en la Blockchain y una lista de los productos ya registrados.

- **ManipulatorManager.js:** Componente que consigue, de la Blockchain, la cantidad de manipuladores que están registrados (Longitud del Array que contiene Manipulators.sol en el contrato ManipFactory.sol). Con esta cantidad, crea tantos componentes ShowManipulator.js como manipuladores están registrados para que puedan ser listados.

- **ShowManipulator.js:** Componente que muestra un manipulador. Puede mostrarlo de dos formas: La primera, con su código QR, que representa su dirección de despliegue, y el índice que ocupa en la lista o id. La segunda, al hacer click sobre este componente, muestra el componente Manipulator.js relativo al manipulador tratado.

- **Manipualtor.js:** Componente que muestra toda la información almacenada en la Blockchain relativa a un manipulador.

- **ItemManager.js:** Componente equivalente a ManipulatorManager.js. Sirve para obtener la cantidad de productos registrados en la Blockchain y generar tantos componentes ShowItem.js como productos haya.

- **ShowItem.js:** Componente equivalente a ShowManipulator.js. Muestra un producto de dos formas distintas: La primera, muestra el código QR referente a su dirección de despliegue y su índice o id. La segunda, que se muestra al clicar en este componente, muestra el componente Item.js referente a este producto.

- **Item.js:** Componente que muestra toda la información almacenada en la Blockchain referente a un producto. A su vez, nos muestra la ruta que a seguido este producto a través del componente ShowRoute.js.

- **ShowRoute.js:** Componente mostrado al pulsar el botón "Mostrar/Ocultar ruta" que se puede encontrar en los componentes Item.js. Este componente muestra tantos componentes Visit.js como hitos en la ruta del producto se hayan seguido.

- **Visit.js:** Componente que muestra toda la información almacenada en la Blockchain. Para mostrar la información del manipulador encargado de ese hito, muestra un componente ShowManipulator.js para representarla.

4.3. PRUEBAS

Para acabar con cualquier desarrollo de una aplicación, se deben implementar pruebas para asegurar su correcto funcionamiento. En este caso, se realizarán pruebas unitarias a cada contrato de Solidity. Estas, se realizarán en JavaScript y se ejecutarán con la ayuda de Truffle. Para la parte de navegación o de las tecnologías cliente usadas, se realizarán pruebas usando el sistema en ejecución. Estas pruebas consistirán simplemente en navegar e interactuar con la aplicación para comprobar que se comporta como debe. Ya que los flujos de navegación son bastante simples, no es necesario probar con más intensidad este apartado.

4.3.1. PRUEBAS UNITARIAS

Estas pruebas se desarrollarán y probarán en un entorno distinto. Para ello, se creará un nuevo directorio o entorno con Truffle, se crearán las pruebas con JavaScript en la carpeta "test/" creada por Truffle y se ejecutarán a través del comando "test" que nos facilita Truffle.

Estas pruebas se encuentran en una nueva rama "Testing" del repositorio de Github antes mencionado:

https://github.com/Jaimedfc/TFG

Con estas pruebas, se busca un correcto funcionamiento de los contratos creados: se crean correctamente, sus métodos funcionan con normalidad.

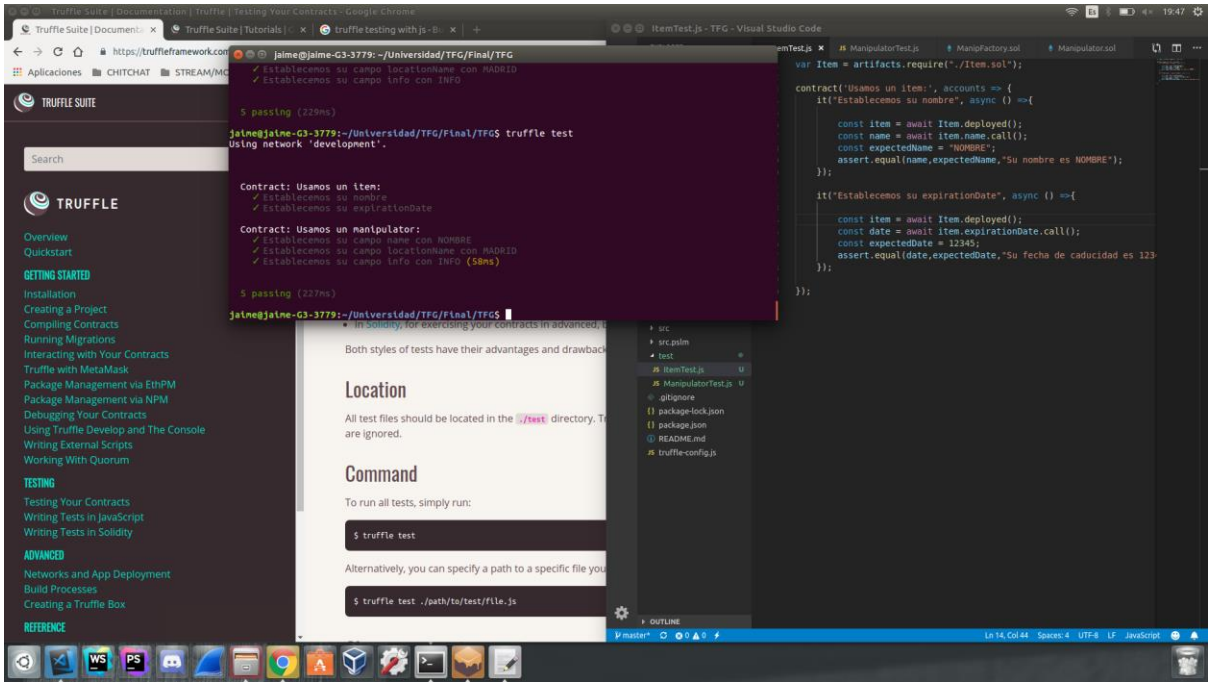

Ilustración 33: Captura de pantalla del resultado obtenido al ejecutar las pruebas unitarias

4.3.2. PRUEBAS DE EJECUCIÓN

Estas pruebas se centrarán en utilizar la aplicación poniéndose en la piel de los distintos actores:

- Como administrador, se quiere registrar manipuladores en la Blockchain de forma correcta y cómoda.
- Como manipulador, se quiere registrar nuevos productos y reflejar en ellos información de la ruta que han seguido.
- Como cliente, se quiere ver de forma sencilla y atractiva toda la información referente a los distintos productos registrados con anterioridad.

A continuación, se exponen las distintas acciones realizadas para llevar a cabo las distintas pruebas:

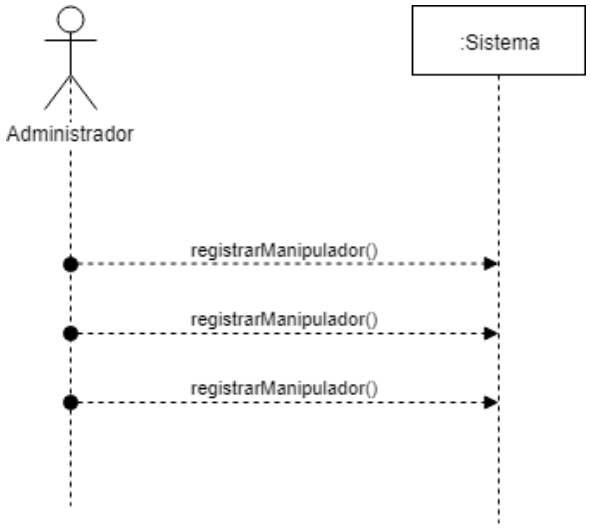

Ilustración 34: Acciones tomadas por el administrador en las pruebas de navegación

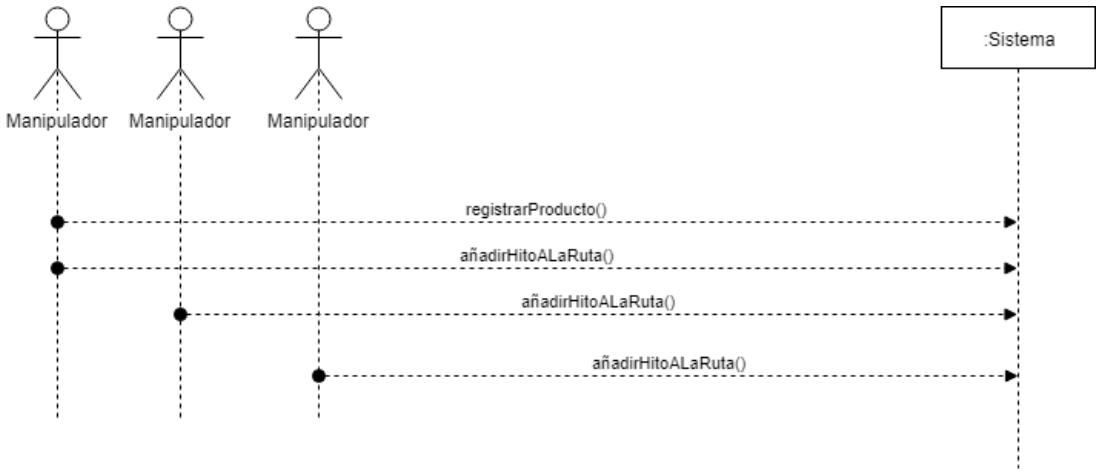

Ilustración 36: Acciones tomadas por los manipuladores previamente registrados

Ilustración 35: Acción realizada por el cliente para comprobar las acciones previamente realizadas

Cabe destacar que cada actor utilizará, sin ser conscientes de ello, distintos contratos para llevar a cabo sus acciones:

- El administrador utilizará ManipFactory para registrar y eliminar manipuladores del sistema. A su vez, también utilizara ItemFactory para eliminar los productos descatalogados o caducados del sistema.
- Los distintos manipuladores, utilizarán ItemFactory para registrar productos en el sistema y luego utilizarán el contrato perteneciente a cada producto para añadir los distintos hitos de sus rutas.
- Los clientes solo utilizaran los distintos contratos pertenecientes a todos los productos del sistema. Simplemente harán uso de los distintos métodos “getters” o de solo lectura de datos.

5. MANTENIMIENTO Y OPERACIÓN

Con el despliegue del servicio o la finalización de la implementación no acaba un servicio software. El mantenimiento es una de las partes que más ingresos genera a las distintas empresas y es una de las partes donde más esfuerzo se invierte. Esta aplicación no va a ser distinta.

El encargado de un primer grado de mantenimiento será el administrador del sistema. Con las distintas funciones de borrado o eliminación de contratos de la Blockchain. Una vez un contrato resulta inservible, ya sea porque un producto ha caducado o porque un manipulador ya no se encuentra vinculado a nuestro sistema, es necesario eliminarlo del sistema, ya que dedicar distintos "listeners" a vigilar contratos inservibles es un gasto de capacidad de computo innecesario.

Por otra parte, la manera en la que se manejan los productos y manipuladores en los contratos factoría (ItemFactory y ManipFactory) no es la forma más eficiente y barata en cuanto a computación. Debido a esto, es necesario un buen mantenimiento por parte del administrador.

El mantenimiento de segundo grado podría llamársele al realizado sobre las distintas cuentas con los fondos para interactuar con la Blockchain. Cada llamada de escritura en la Blockchain cuesta un dinero virtual, "ethers", el cual puede llegar a gastarse. Si tal caso se diese, se necesitaría una recarga de fondos para poder seguir usando el sistema con normalidad. En este desarrollo se ha estado usando una red de pruebas perteneciente a Ethereum, la cual al ser arrancada crea diez cuentas distintas llenas de fondos para realizar pruebas, para este desarrollo no ha sido necesario "gastar dinero". Por este motivo, no se ha tenido en cuenta el gasto de cada llamada, es decir, este apartado no se ha realizado eficientemente. En un desarrollo profesional, con una red pública de Blockchain, será necesario un cierto gasto con cada llamada de escritura. Será necesario pulir este apartado en este caso. Si usamos una red privada, este no sería un problema ya que podríamos recargar fondos a placer.

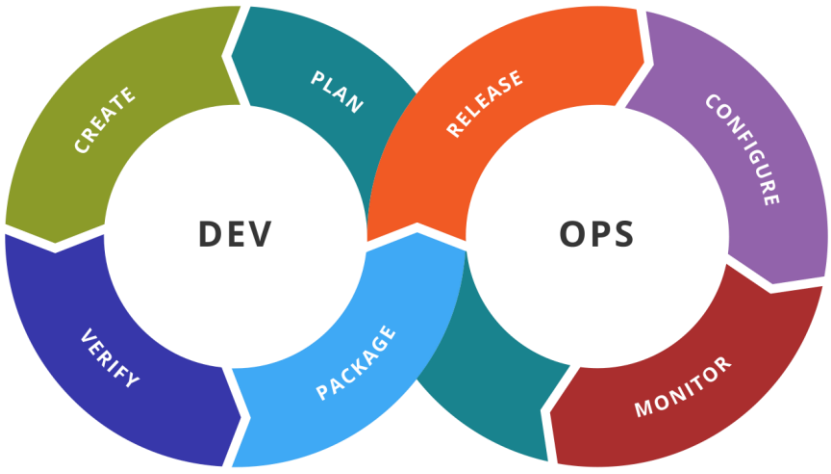

Ilustración 37: Flujo de trabajo de los equipos de desarrollo y operación

6. CONCLUSIONES Y LÍNEAS FUTURAS

Para finalizar esta memoria, se recogerá, en estos apartados, las distintas conclusiones a las que se han llegado, los objetivos que se han conseguido cumplir y un pequeño resumen de mejoras a realizar para un posible desarrollo profesional del sistema.

6.1. CONCLUSIONES

Para comenzar con las conclusiones, me gustaría comentar el tiempo que ha llevado este trabajo. Blockchain, con los conocimientos adquiridos a lo largo del grado, no tiene una barrera excesivamente alta de entrada. Solo se necesita familiarizarse con ciertos términos nuevos, tales como Smart Contracts, mineros… Este trabajo me ha llevado un total de 4 meses, en los que trabajaba cada fin de semana o hueco que tenía entre semana. Cada día de trabajo suponía un mínimo de 2 horas. 3 de esos 4 meses los pasé aprendiendo acerca de Blockchain y acerca de cómo realizar DApps. La realización de la primera versión funcional se llevó a cabo durante el mes de Mayo. Con esto quiero decir que, tres cuartos del tiempo se invirtieron en aprender acerca de las tecnologías, y un cuarto se invirtió en el desarrollo.

También quiero decir que no me arrepiento de la elección que hice acerca de este TFG. Me ha gustado llevarlo a cabo y me ha gustado ver de lo que soy capaz de hacer estando solo. Imagino que estando en un equipo multidisciplinar, este trabajo habría podido realizarse en la mitad de tiempo y podría haber alcanzado una escala mucho mayor.

6.2. OBJETIVOS CUMPLIDOS

Se ha conseguido desarrollar una DApp funcional, aunque a una escala menor que la que desarrollaría una empresa en un entorno real. Con esto se han podido ver las fortalezas y puntos débiles de las tecnologías usadas, sobre todo en el lenguaje de programación de los contratos inteligentes Solidity. Por otra parte, esta tecnología se encuentra en un estado de desarrollo joven y aún hay mucho espacio para mejorar aspectos.

En mayor profundidad, también se han visto las facultades de Ethereum y Blockchain. Sus Smart Contracts tienen un potencial enorme en aplicaciones que involucren terceras partes. Por ejemplo, la compraventa de una vivienda implica la necesidad de un notario. Con una aplicación desarrollada sobre Blockchain, este notario desaparecería y el nivel de confianza no disminuiría.

6.3. LÍNEAS FUTURAS

Como se ha visto en varias ocasiones a lo largo de esta memoria, la parte que se ha desarrollado no es el final de este trabajo. Si se quiere desplegar y poner en funcionamiento en un entorno profesional, hay que hacer varios retoques a lo hecho hasta el momento.

6.3.1. CAMBIOS EN LOS SMART CONTRACTS

Los Smart Contracts utilizados hasta ahora están bien para un desarrollo teórico o didáctico, sin embargo, no son adecuados para un despliegue profesional. Necesitan lo siguiente:

- Aumentar su **seguridad** e introducir permisionado. Actualmente, cualquier persona ajena a la red de Blockchain puede hacer llamadas a los contratos. Esto es impensable para un entorno profesional. Se solucionaría introduciendo un campo extra en cada contrato con las

direcciones que tienen permitido modificar o crear contratos. Por ejemplo, los contratos relacionados con los manipuladores solo pueden ser modificados, eliminados o creados por administradores; los contratos referentes a productos solo pueden ser creados y modificados por manipuladores y eliminados por administradores; y todos los contratos pueden ser leídos por cualquiera. Por otra parte, hay que ser consciente de que, por regla general, el código de los Smart Contracts va a estar disponible en la red. Conviene tomar ciertas medidas y llevar a cabo buenas prácticas para evitar ataques o usos maliciosos de estos contratos [31]. También hay que señalar que, aunque un contrato no tenga ningún bug, existe la posibilidad de que el compilador lo cree. En definitiva, nunca podemos estar 100% seguros de que un contrato, o artefacto software, está libre de bugs o código inseguro.

- En los contratos se usan varios algoritmos de búsqueda y guardado muy poco eficientes. En un futuro, convendría usar algoritmos más eficientes para ello.
- Incluir un método en el contrato que maneja los productos, ItemFactory.sol, que, pasándole un argumento que sea la fecha de hoy, elimine todos los contratos que hagan referencia a productos caducados.

6.3.2. CAMBIOS EN LA ARQUITECTURA

En apartados anteriores, se han mostrado dos arquitecturas diferentes, aunque funcionalmente son iguales. Para un despliegue profesional, se necesitarán cambios en:

- **La red de Blockchain:** Es necesario el uso de una red privada de Ethereum para nuestro sistema. La red de Ganache es útil, aunque como la propia definición de Ganache dice, es solo para pruebas o para la fase de desarrollo, no es tan potente o segura para grandes despliegues. Esta red podría desplegarse a través de Amazon Web Services [32] (AWS), para ello se necesitaría alquilar un gran número de máquinas virtuales que actuarán como nodos de nuestra red. Estos nodos podrían tener los siguientes roles [33]:
    - Bootnode: Estos nodos actuarían como Discovery Service. Permitirá a los demás nodos agregarse automáticamente a la red y preguntar por los demás nodos.
    - Nodo minero: Nodo encargado de generar bloques a través del algoritmo de PoW (Proof of Work) [34]. Al ser nuestra red Blockchain, una red privada, no necesitaríamos este algoritmo tan pesado, nos valdría con un PoA (Proof of Authority), el cual se basa en que, si tenemos la autoridad de enviar transacciones a la red, podremos crear los bloques. El que no esté autorizado, alguien ajeno a nuestra red, no podrá generar bloques.
    - Nodo de trabajo: Nodo con el que nos comunicaremos para interactuar con los Smart Contracts.

  Cabe destacar que, a mayor número de nodos o máquinas, mayor seguridad tendrá nuestra red.

- **Servidores que contienen la aplicación web:** En el desarrollo solo se usa una máquina para correr la aplicación. En un entorno más avanzado será necesario tener más máquinas que puedan atender a un gran número de llamadas. También sería conveniente establecer “firewalls” y “load balancers” para mayor disponibilidad del servicio.

6.3.3. CAMBIOS EN LA APLICACIÓN WEB

En la aplicación Web actual, se puede acceder a cualquier vista sin necesidad de autentificarse. En un futuro, se necesitará establecer una forma de autentificación, ya sea mediante un LogIn o cualquier otra forma, o crear distintas aplicaciones, una para cada actor. Esto último sería lo más conveniente ya que para los clientes sería conveniente desarrollar una aplicación móvil con funciones de cámara para escanear los códigos QR asociados a cada producto. Esta aplicación se podría hacer con React-Native con mucha facilidad ya que solo conllevaría cambiar partes de sintaxis del código actual ya que, las partes funcionales no cambian entre estas tecnologías (React y React-Native).

Hay detalles que habría que mejorar también de la aplicación. Uno de ellos es el saneamiento de los formularios o comprobación de los datos introducidos en los campos. Estos datos podrían introducirse incorrectamente, por ejemplo, introducir texto donde hay que introducir valores numéricos. También podría inyectarse código malicioso a través de estos formularios.

Habría que modificar el aspecto estético de la Web, adaptarla a la imagen corporativa que vaya a desarrollar o usar esta aplicación. Relativo a la estética, habría que usar una imagen distinta para cada producto y manipulador, ya que, en la aplicación actual se usa la misma imagen para cada tipo de producto.

6.3.4. DISPOSITIVOS A USAR POR LOS MANIPULADORES

En este sistema, sería conveniente que los futuros manipuladores usasen dispositivos especializados y configurados por la empresa que lleve a cabo el despliegue de la aplicación, tales como, escáneres de códigos QR con “tablets” u ordenadores para introducir los valores deseados en la Blockchain.

Por otra parte, los manipuladores, al igual que los clientes, requerirán de una aplicación móvil o Web especializada para que desarrollen su actividad cómodamente.

6.3.5. ESCALABILIDAD

En este apartado se tratará el problema de la escalabilidad de nuestros Smart Contracts.

Existe una razón esencial, por la cual no usamos la red pública de Ethereum, y esa es la escalabilidad. Nuestra aplicación generará una gran cantidad de contratos (un contrato por cada producto individual), así como transacciones en general. La red pública de Ethereum genera un bloque cada 14 segundos. Esta limitación nos complica muchos aspectos de nuestra aplicación. Por otro lado, tendríamos que gastar “ethers” o criptomonedas en cada transacción.

Por estos motivos, se ha elegido como solución utilizar, o construir, una red privada con la tecnología de Ethereum.

7. BIBLIOGRAFÍA

[1] «Speeding Up The Widepipe: Secure and Fast Hashing,» [En línea]. Available: https://eprint.iacr.org/2010/193.

[2] «Bitcoin Wiki,» [En línea]. Available: https://en.bitcoin.it/wiki/Main_Page.

[3] «Ethereum Website,» [En línea]. Available: https://www.ethereum.org/.

[4] «Hyperledger Website,» [En línea]. Available: https://www.hyperledger.org/.

[5] «Solidity Documentation - Introduction to Smart Contracts,» [En línea]. Available: https://solidity.readthedocs.io/en/v0.5.9/introduction-to-smart-contracts.html.

[6] «Public versus private Blockchain - White Paper,» [En línea]. Available: https://bitfury.com/content/downloads/public-vs-private-pt1-1.pdf.

[7] «Truffle Suite - Ganache,» [En línea]. Available: https://www.trufflesuite.com/ganache.

[8] «Truffle Suite Website,» [En línea]. Available: https://www.trufflesuite.com/.

[9] «Mozilla Developer Website - JavaScript Documentation,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

[10] «ReactJS Website,» [En línea]. Available: https://reactjs.org/.

[11] «Truffle Suite - Drizzle,» [En línea]. Available: https://www.trufflesuite.com/drizzle.

[12] «Web3JS Documentation,» [En línea]. Available: https://web3js.readthedocs.io/en/1.0/index.html.

[13] «JSON Website,» [En línea]. Available: https://www.json.org/.

[14] «JSON-RPC Specification,» [En línea]. Available: https://www.jsonrpc.org/specification.

[15] «Redux Documentation,» [En línea]. Available: https://es.redux.js.org/.

[16] «Solicity Documentation v0.5.9,» [En línea]. Available: https://solidity.readthedocs.io/en/v0.5.9/.

[17] «NodeJS Website,» [En línea]. Available: https://nodejs.org/es/.

[18] «NPM Website,» [En línea]. Available: https://www.npmjs.com/.

[19] «QRcode.react Library Documentation,» [En línea]. Available: https://www.npmjs.com/package/qrcode.react.

[20] «Reactstrap Documentation,» [En línea]. Available: https://reactstrap.github.io/.

[21] «Bootstrap 4 - Documentation,» [En línea]. Available: https://getbootstrap.com/docs/4.0/getting-started/introduction/.

[22] «Google Maps - Developer Guide,» [En línea]. Available: https://developers.google.com/maps/documentation/urls/guide.

[23] «SublimeText Website,» [En línea]. Available: https://www.sublimetext.com/.

[24] «Visual Studio Code Website,» [En línea]. Available: https://code.visualstudio.com/.

[25] «UML Diagram types guide,» [En línea]. Available: https://creately.com/blog/diagrams/uml-

diagram-types-examples/.

[26] «ECSS-E-ST-40C Specification Website,» [En línea]. Available: https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/.

[27] «MetaMask Website,» [En línea]. Available: https://metamask.io/.

[28] «React Native Documentation,» [En línea]. Available: https://facebook.github.io/react-native/.

[29] «YouTube - Carrefour tecnología Blockchain,» [En línea]. Available: https://www.youtube.com/watch?v=7Zey82mBTg0.

[30] «Truffle Suite - Drizzle Documentation - Contract Interaction,» [En línea]. Available: https://www.trufflesuite.com/docs/drizzle/getting-started/contract-interaction.

[31] «GIT Website,» [En línea]. Available: https://git-scm.com/.

[32] «Ciclo de vida de un componente React,» [En línea]. Available: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/.

[33] «Solidity Documentation - Security Considerations,» [En línea]. Available: https://solidity.readthedocs.io/en/v0.5.9/security-considerations.html.

[34] «Amazon Web Services Website,» [En línea]. Available: https://aws.amazon.com/es/.

[35] «Tutorial para desplegar red de blockchain con AWS,» [En línea]. Available: https://enmilocalfunciona.io/despliegue-de-una-red-blockchain-privada-sobre-aws/.

[36] «Bitcoin Wiki - Proof of Work,» [En línea]. Available: https://en.bitcoin.it/wiki/Proof_of_work.

[37] «COIT - El Perfil del Ingeniero de Teleomunicaciones,» [En línea]. Available: https://www.coit.es/sites/default/files/informes/pdf/mapa-del-estudiante-de-ingenieria-de-telecomunicacion_0.pdf#pdfjs.action=download.

[38] «AWS - Precios de instancias EC2,» [En línea]. Available: https://aws.amazon.com/es/ec2/pricing/reserved-instances/pricing/.

[39] «Truffle Suite - Getting started with Drizzle Tutorial,» [En línea]. Available: https://www.trufflesuite.com/tutorials/getting-started-with-drizzle-and-react.

ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

A.1 INTRODUCCIÓN

En la actualidad, es cada vez más habitual que las empresas publiquen y enseñen datos acerca de sus ejercicios o acciones tomadas para aportar más transparencia y aumentar la confianza con las personas, ya sean clientes, accionistas…

Este trabajo busca desarrollar una aplicación para trazar productos, que todo tipo de persona puede comprar en un supermercado. Esto busca aumentar la confianza con sus clientes.

Debido a estos aspectos, esta aplicación tendría un gran impacto en varios temas. En este anexo, se discutirán los impactos sociales, económicos, éticos y ambientales que está aplicación puede acarrear.

A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Uno de los impactos más relevantes, e implícito en todas las aplicaciones desarrolladas sobre Blockchain es el ahorro económico que implican las terceras partes contratadas para aportar confianza. En nuestro caso, trazando mercancías, esto no supondrá un gran ahorro, pero en futuras aplicaciones, dedicadas a votaciones y elecciones, por ejemplo, este ahorro podría suponer mucho.

Otro impacto recaerá en los aspectos sociales y éticos, que siempre suelen ir de la mano. Esta aplicación, aportará muchos datos acerca de la crianza de animales en granja o el crecimiento de plantas de cultivo. Este sistema desarrollado puede llegar a concienciar a muchas personas, y estas a su vez pueden llegar a exigir mejoras en los distintos lugares de producción, ya sea en lo referente a los propios productos, o a los empleados involucrados con ellos. En casos extremos, se expondrían casos de explotación laboral.

En el ámbito legal, este sistema no supondrá ningún problema.

En cuanto al ámbito ambiental, un impacto considerable sería el gasto de luz que implicará el funcionamiento de las numerosas máquinas en la nube contratadas o alquiladas para crear nuestra red privada de Blockchain. Por otra parte, se podrían exponer irregularidades medioambientales en la línea de producción y transportes de los distintos productos. Por ejemplo, si se decide mostrar datos acerca de los herbicidas usados en los cultivos, se podría exponer a algunos agricultores que estén usando productos prohibidos.

A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

Como se ha comentado en el apartado anterior, el aspecto ético y social sería de los aspectos en los que esta aplicación influiría en mayor parte. Esta aplicación mostrará de forma accesible mucha información relevante con la cría de animales, así como el cultivo de vegetales y frutas. Gran parte de la población no suele interesarse en estos aspectos, en parte por el difícil acceso a este tipo de información. Pero una vez existe una forma fácil de acceder a este tipo de datos, la gente comenzará a preocuparse por estos aspectos. Esto aportará muchos beneficios ya que la gente exigirá mayor calidad en las etapas tempranas en la cadena de producción.

A.4 CONCLUSIONES

Como se ha visto en los anteriores apartados, esta aplicación no aportará ningún efecto negativo a los distintos aspectos tratados: sociales, económicos, éticos y ambientales. Lo que es más, en algunos aspectos, podría llegar a ser beneficioso para la sociedad actual.

ANEXO B: PRESUPUESTO ECONÓMICO

Para desarrollar el presupuesto económico, se tendrán en cuenta varias suposiciones:

- El sueldo medio de un ingeniero de telecomunicaciones es de 52.711€ al año según el COIT [35], repartidos en 14 pagas al año trabajando 20 días al mes (4 semanas), 8 horas diarias.
- La duración del desarrollo será de 312 horas, el equivalente de 12 créditos.
- El software no implica gasto alguno pues se usa software de código abierto o libre.
- Se alquilarán 3 máquinas virtuales (aunque en un desarrollo profesional se necesitarían muchas más) durante 1 año.
- El alquiler de una máquina EC2 t2.micro en AWS cuesta 0.012€/hora (0,0132 $/hora) en la región de UE Londres [36].

Tras estas consideraciones, llegamos al siguiente presupuesto:

COSTE DE MANO DE OBRA (coste directo)	**Horas**	**Precio/hora**	**Total**
	312	23,53 €	**7.341,36 €**

COSTE DE RECURSOS MATERIALES (coste directo)	**Precio de compra**	**Uso en meses**	**Amortización (en años)**	**Total**
Ordenador personal (Software incluido)	1.000,00 €	6	5	100,00 €
Alquiler 3 máquinas virtuales AWS EC2 t2.micro durante 1 año	315,36 €	12	1	315,36 €
Otro equipamiento				

COSTE TOTAL DE RECURSOS MATERIALES	**415,36 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**1.163,51 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**535,21 €**

SUBTOTAL PRESUPUESTO		**9.455,44 €**
IVA APLICABLE	21%	**1.985,64 €**

TOTAL PRESUPUESTO	**11.441,08 €**

El presupuesto resultante, 11.441,08 € será una cota inferior ya que, en un auténtico despliegue, se necesitarían más de 3 máquinas y seguramente se necesitarían máquinas más potentes que las usadas en el presupuesto. También será necesario tener en cuenta los costes de operación y mantenimiento del sistema.

ANEXO C: MANUAL DEL USUARIO

En este anexo se explicarán los pasos necesarios para la descarga, instalación y puesta en marcha del proyecto desarrollado en el repositorio https://github.com/Jaimedfc/TFG .

C.1 INSTALACIÓN Y EJECUCIÓN

En primer lugar, será necesario tener instaladas las siguientes herramientas (preferiblemente últimas versiones estables):

- Git.
- NodeJS y npm.
- Truffle.
- Ganache (versión de línea de comando, ganache-cli o versión con interfaz gráfica).

En primer lugar, descargamos o clonamos el proyecto en nuestra máquina, lo podemos hacer con el siguiente comando:

```
git clone https://github.com/Jaimedfc/TFG
```

Ahora, nos cambiamos al nuevo directorio TFG con:

```
cd TFG
```

Con la ayuda de npm, instalamos todas las dependencias que necesitaremos (se encuentran descritas en el archivo package.json) con el siguiente comando:

```
npm install
```

Una vez instaladas todas las dependencias, pasamos a usar el framework de Truffle compilando los contratos que usaremos:

```
truffle compile --all
```

Este comando crea una nueva carpeta dentro de src con los contratos ya compilados dentro.

Durante los siguientes pasos, necesitaremos tener en funcionamiento nuestra red privada personal de Blockchain Ganache. Simplemente abriendo la interfaz gráfica y haciendo click en “Quickstart”. Debemos asegurarnos de que la dirección de despliegue de la red es 127.0.0.1 y puerto 7545 en los ajustes de Ganache (Si se usasen otras direcciones o puertos, conviene modificar el archivo truffle-config.js).

Ahora que tenemos Ganache funcionando, podemos migrar o desplegar los contratos en nuestra red con el siguiente comando:

```
truffle migrate --reset
```

La opción --reset no es necesaria la primera vez que migramos los contratos, pero las siguientes veces será necesario.

Ya podemos ejecutar nuestra aplicación, podemos hacerlo con:

```
npm start
```

Se nos abrirá una ventana o pestaña de nuestro navegador con la aplicación. En caso de no abrirse automáticamente, solo tendremos que acudir a esta dirección en nuestro navegador:

http://localhost:3000/

C.1 POSIBLES FALLOS O PROBLEMAS

Si tenemos instalada y habilitada en nuestros navegadores Chrome la extensión “Metamask”, esta, entrará en conflicto con Ganache y afectará al correcto funcionamiento de nuestra aplicación. Conviene desactivar esta extensión durante la ejecución de la aplicación.

ANEXO D: TUTORIAL O PRIMEROS PASOS PARA CREAR UNA DAPP

El siguiente desarrollo está disponible en mi perfil de Github:

https://github.com/Jaimedfc/BlockChain-SmartContractSencillo

D1. REQUISITOS DEL SISTEMA

Se necesitarán las siguientes herramientas en su última versión estable:

- NodeJS y npm.
- Truffle.
- Ganache (versión de línea de comando, ganache-cli o versión con interfaz gráfica).

D2. TUTORIAL DE TRUFFLE-FRAMEWORKS

Para comenzar, es recomendable realizar y seguir el tutorial que nos proporciona Truffle-Frameworks [37]. Es una buena toma de contacto ya que ofrece explicaciones sencillas de todos los aspectos involucrados en el desarrollo.

Una vez completado este primer tutorial, tendremos lo siguiente:

- Un contrato inteligente con un único atributo de tipo “String” con sus respectivos “getters” y “setters” o métodos para gestionar este atributo.
- Una aplicación web sencilla con la que podremos presentar el atributo de nuestro contrato y cambiarlo a placer.

D3. CONTRATO CON UNA LISTA

Nuestro siguiente objetivo es evolucionar nuestro contrato y aplicación. En vez de poseer un único atributo, pasaremos a tener un conjunto de ellos almacenados en un “Array”. Nuestra aplicación deberá presentarnos todos los “Strings” almacenados y nos deberá permitir cambiarlos.

Para esto, en nuestro contrato necesitamos cambiar el tipo del atributo a un “Array” de “Strings”. También debemos crear un nuevo método que nos tendrá que devolver la longitud de este “Array”, ya que la necesitaremos más adelante. Nuestro contrato quedaría así:

```
pragma solidity ^0.5.0;


contract Factory {
  string[] public items = ["Hola","Adios"];
  event Tic(string msg, uint);

  function set(string memory x) public{

        items.push(x);
        emit Tic("Actualizado",items.length);


  }

  function long() public view returns (uint){

        return items.length;
  }

}
```

Ilustración 38: Código de un contrato Factoría

Ahora necesitamos retocar nuestros componentes React. Antes, solo era necesario un componente, ahora, necesitaremos dos. La razón es sencilla: Al tener una lista de elementos, necesitamos crear un observador para cada uno de estos elementos, no nos vale con crear uno único para la lista. Aquí, es donde entra nuestro método para obtener la longitud del "Array".

Tendremos dos componentes, un padre, y un hijo que cuelgue del padre. En el padre, crearemos un observador para el método de obtención de longitud, para luego, en el mismo componente padre recuperar este valor y pasárselo al componente hijo como "prop". Una vez en el componente hijo, crearemos tantos observadores como nuestro componente padre nos diga (Esto en el método componentDidMount() y en el método componentDidUpdate()). Ahora solo tenemos que obtener los valores de la lista tal y como hicimos en el contrato anterior. El componentDidUpdate() quedaría así:

```
componentDidUpdate(prevProps, prevState, snapshot) {

const l1 = prevProps.long || 0;
const l2 = this.props.long;
const { drizzle } = this.props;
const contract = drizzle.contracts.Factory;
var dataKey = this.state.dataKey;
if (l1 < l2) {
    for (var i=l1; i<l2; i++){
        dataKey[i] = contract.methods["items"].cacheCall(i);
    }
    this.setState({ dataKey });
}


}
```

Ilustración 39: Código del método componentDidUpdate() en un componente React

Este método es llamado cada vez que se cambia o actualiza el componente. En él, programamos que, cada vez que la longitud pasada en las props (la cantidad de elementos) aumenta, queremos observar estos nuevos elementos.

D4. CONTRATO FACTORÍA

Como último paso, nos centraremos en crear un contrato orquestador de contratos, un contrato capaz de crear contratos dinámicamente y administrarlos. Este despliegue se encuentra aquí:

https://github.com/Jaimedfc/BlockChain-SmartContractSencillo/tree/paso2-AccederContratoDesdeOtro

Este contrato factoría debe ser capaz de lo siguiente:

- Crear contratos.
- Almacenar los contratos creados.
- Llamar a métodos de estos contratos.

La parte Back-End (Solidity) de este apartado es bastante sencilla si se tiene experiencia en otro tipo de lenguajes orientado a objetos. En el contrato factoría solo se debe importar el código de los contratos que se quieren crear y escribir un método constructor en estos. Para importar, se debe escribir lo siguiente justo después de la descripción de la versión de solidity a usar:

```
import './Item.sol';
```

El contrato Item.sol sería algo así:

```
pragma solidity ^0.5.0;

contract Item {


        string public good;


        constructor(string memory x) public {
                good = x;

        }


        function setGood(string memory x) public {

                good = x;

        }

}
```

Ilustración 40: Código del contrato manejado por una Factoría de contratos

Con esto acabaríamos con la parte Back-End. La parte del Front-End o React/Drizzle es muy similar a nuestro apartado anterior en el que tratábamos un conjunto de Strings. La única peculiaridad encontramos es el registro de nuestros nuevos contratos en Drizzle, para así, poder usarlos a través de nuestra aplicación. En la siguiente referencia encontramos qué debemos hacer en el método componentDidMount() de los componentes que representarán cada nuevo contrato [38].

Nuestro componentDidMount() quedaría así:

```
    componentDidMount() {

        console.log("==== COMPONENTE ItemContract MONTADO ============", this.props.address );

        const { drizzle } = this.props;

        const json = require('../contracts/Item.json');

          const contractConfig = {
             contractName: this.props.address,
             web3Contract: new drizzle.web3.eth.Contract(json.abi, this.props.address)
          };

          drizzle.addContract(contractConfig, []);
    }
```

Ilustración 41: Código del método componentDidMount() de un componente React

Con la última sentencia, le decimos a Drizzle que existe un nuevo contrato, con una estructura y dirección definidas, que queremos vigilar u observar.

Una vez completado y asimilado este paso, ya seremos capaces de construir servicios o DApps sobre Blockchain con cierto nivel de complejidad.