ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería Informática - Ingeniería del Software

DOCTRACK: Herramienta para Entrega y Seguimiento de Documentos con Blockchain

Realizado por
Ignacio Navarro Blázquez

Dirigido por
Ángel Jesús Varela Vaca

Departamento
Lenguajes y Sistemas Informáticos

Sevilla, (07/2022)

Resumen

Este proyecto tiene como objetivo la creación de una *dApp* o aplicación descentralizada, que provee a los usuarios de un sistema de entrega y gestión de documentos a empresas mediante la tecnología blockchain. Para ello, se ha realizado una aplicación web de página única en ReactJS, y varios contratos inteligentes en Solidity, desplegados en una red de prueba de la blockchain de Ethereum.
Palabras clave: Documentos, Gestión, Blockchain, Contratos inteligentes

This project aims to create a *dApp* or decentralized application to provide users a delivery and management system through blockchain technology. For this purpose, a single web application has been made with ReactJS, and various smart contracts have been written in Solidity and deployed in an Ethereum testnet.
Keywords: Documents, Management, Blockchain, Smart Contracts

Agradecimientos

En primer lugar, agradecer a mi familia por el apoyo incondicional que me han brindado a lo largo de toda la carrera y del proceso de creación de este proyecto, con el que finaliza una etapa de mi vida. En especial a mis padres, que siempre me apoyaron en todo lo que he realizado.

También dar las gracias a todos los amigos y compañeros que he conocido en la carrera, y a los que ya estaban, por hacer que estos cuatro años se hayan pasado volando. Especialmente a Fernando y a Nicolás, pues son los mejores amigos que uno podría desear, y me han apoyado, aconsejado y ayudado en todo momento.

Por último, a mi tutor, Ángel Jesús Varela, por haberme guiado en todo el proceso y facilitarme el proceso sabiendo que estaba en Alemania.

Índice general

Índice de tablas

Índice de figuras

Índice de código

CAPÍTULO 1

Introducción

Desde hace varias décadas, la comunicación entre las administraciones públicas o empresas privadas y los usuarios o clientes no ha sufrido ninguna gran modificación, donde la mensajería y entrega de documentos pueden realizarse por medios físicos o digitales como por ejemplo los registros civiles o generales de las administraciones públicas.

Al principio, sólo se podía hacer en papel o por correo certificado, y más tarde, con el nacimiento de la firma digital [5], cada vez más empresas se adaptaron al nuevo medio, pero este, sólo lleva un registro del documento, sin hacer seguimiento de las personas o entidades por las que pasa una vez entregado.

Además, los tiempos de revisión o respuesta se dilatan ampliamente, en la mayoría de los casos, sin motivo justificado. A la vez que no comunican al usuario el estado en el que se encuentran, dónde ni con quién, dejando una incertidumbre a este, por la falta de información que existe en el proceso, hasta el momento en el que finaliza y se comunica al usuario.

No sólo existe un problema temporal, también existe el problema de la entrega de documentos oficiales o importantes, los cuales, en algunos casos, requieren entrega física para confirmar la validez del documento y la identidad de la persona que los entrega o del solicitante.

Según un estudio de Zendesk [26], las tres claves para satisfacer al cliente de manera correcta son:

- Conocer a los usuarios.
- Reducir los tiempos de espera.
- Brindar una buena experiencia digital.

En esta era donde cada vez más, está todo digitalizado, surge la necesidad de crear una herramienta que pueda solucionar estos problemas,

siendo así, simple, confiable, segura y eficaz.

1.1– Contexto y motivación

La existencia de herramientas de envío y registro de documentos adecuadas o anticuadas, o incluso inexistencia de soluciones, generó la necesidad de crear un sistema que cubra esos problemas. Por ello, se buscó crear una solución que tuviera las siguientes características:

- Seguridad.
- Trazabilidad.
- Inmutabilidad.
- Transparencia.
- Veracidad.

Las cuales son recogidas entre las características de la blockchain [20].

1.1.1. Blockchain

La blockchain o cadena de bloques es un libro de contabilidad digital, distribuido e inmutable que registra datos, en cada transacción, de cualquier tipo. Se han mencionado previamente varias de sus características, que son las buscadas por nuestra solución, por ello se decidió usar esta tecnología.

Al oír blockchain, las personas lo relacionan con Bitcoin [18] o las criptomonedas [7], las cuales usan su tecnología y con gran auge en los últimos años, especialmente entre los jóvenes [6], o incluso NFTs o Token No Fungibles [3]. Pero las aplicaciones de esta tecnología son infinitas. Una base de datos pública y transparente, segura que no sea editable una vez se completa la transacción es muy apropiada para problemas de la actualidad.

La seguridad de una blockchain es muy alta, de manera que no se puede cambiar ningún bloque a menos que el 51 % de los nodos estén de acuerdo, es lo que se conoce como Ataque del 51 % [2].

Existen diferentes tipos de blockchain según como operan y su transparencia. Estos son:

- Pública: son completamente transparentes, y cualquier persona puede montar un nodo en ella, cumpliendo los requisitos establecidos por la blockchain. Bitcoin, Ethereum [11], Cardano o Rose son ejemplos de este tipo de blockchain.

- Privada: Mayor rendimiento, sin comisiones, pero los nodos están centralizados y suelen tener un menor número, lo que lo hace un objetivo para el ataque del 51 % mencionado anteriormente. La solución más conocida es HyperLedger.

- Híbrida: Es una mezcla entre la blockchain pública y privada.

Cada blockchain está programada de una manera, y por tanto, se interactúa de manera diferente. Tras el lanzamiento de la máquina virtual de Ethereum EVM [22], varias blockchain se adaptaron a este sistema y son compatibles con los smart contracts o contratos inteligentes de Ethereum, pero también existen otros tipos como los llamados Scripts de Bitcoin [1]. En nuestra solución usaremos estos contratos puesto que nos ofrecen gran variedad de formas de interactuar con el usuario.

Para interactuar con los contratos inteligentes, es necesario de un proveedor, que nos ofrece una billetera o cartera llamada wallet, la cual contiene nuestros criptoactivos en una dirección única, que nos permiten interactuar pagando una pequeña tarifa por transacción a la blockchain. Existen billeteras frías y calientes. Las billeteras frías son las que están contenidas en un dispositivo hardware como un pendrive, un ejemplo es Ledger. Las calientes, por el contrario, se contienen dentro del dispositivo y tienen acceso directo a la red, como puede ser Metamask [16].

1.1.2. Contratos inteligentes

Los contratos inteligentes o smart contracts son programas almacenados en la blockchain que se ejecutan bajo condiciones predeterminadas de manera autónoma y automática, eliminando intermediarios. Gracias a ello, los contratos se pueden realizar con fines, por ejemplo, financieros, fiscales o de prácticamente cualquier carácter que requiera de un intermediario.

En la red de Ethereum, estos contratos se escriben en Solidity y ejecutados en la EVM [12]. Solidity es un lenguaje de programación de alto nivel, orientado a contratos, con una sintaxis similar a la de JavaScript. Fue creado en 2014 por lo que ahora es la Fundación Ethereum [9] y que se encuentra en su versión 0.8.16.

Puesto que Solidity es el lenguaje de contratos inteligentes más popular y usado, aunque no el único, se ha decidido programar los contratos inteligentes en una red que desplegaremos basada en Ethereum, y con contratos programados en solidity.

Podemos observar en la figura 1.1 un ejemplo de contrato inteligente. Vemos como se declara la versión de Solidity al principio del código, algo obligatorio para cada archivo. En la línea 10, comienza la declaración del contrato con la palabra clave **contract**. Tras las llaves, se declaran las variables, en este caso, number que contendrá un número entero sin signo de 256 bits. La primera función, **storage**, es la encargada de guardar ese

número en la blockchain y lleva la etiqueta public, por lo que cualquier contrato podrá llamar a esa función. Por último, tenemos la función **retrieve**, que además de public tiene otras etiquetas, en este caso, view, que nos dice que es una petición y que no se va a escribir nada en la blockchain, y returns, el cual nos dice que devolverá un tipo que está seguidamente de esta palabra entre paréntesis, en este caso uint256.

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
 */
contract Storage {

    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Figura 1.1: Ejemplo de Smart Contract.

1.2– Objetivo

El objetivo de este trabajo de fin de grado consiste en el desarrollo de una dApp o aplicación descentralizada que permita a usuarios la entrega y trazabilidad de documentos mediante un smart contract y la tecnología blockchain, ofreciendo así solución a los problemas tradicionales de un registro como son el seguimiento del estado del documento de manera transparente, segura, trazable, y fiable. También se establecen los objetivos para que el estudio se dé por finalizado:

- Investigación sobre la tecnología blockchain en la mensajería y entrega de documentos.
- Implementación de una solución en una red blockchain con las siguientes funcionalidades:
  - Registrar usuarios y empresas.
  - Añadir y editar información tras el registro.
  - Ver información sobre las empresas.
  - Envío de documentos.

- Ver el estado de un documento tras su envío.
- Ver las entidades o usuarios que tienen acceso al documento.
- Guardar un registro de los documentos enviados.

CAPÍTULO 2

Planificación y costes

La planificación y el desarrollo seguidas en el proyecto se corresponde al uso de las Metodologías Ágiles, específicamente, se ha usado el proceso ágil de SCRUM [21]. Estas metodologías proporcionan un estilo de producción de software útil de manera rápida, iterativa e incremental, optimizando la productividad y ayudando a controlar los riesgos de manera más sencilla y segura.

Puesto que el desarrollo es realizado por un único integrante, este asumirá todos los roles del proyecto, tanto desarrollador como roles directivos. La planificación temporal se dividirá en sprints de larga duración, que dividen al proyecto en fases, y las daily scrum pasarán a ser cada 15 días, completando un sprint tras una sesión de review y retrospectiva con el tutor.

2.1– Análisis temporal

El análisis temporal del proyecto consta de una tabla con una estimación que se hizo al inicio del proyecto de las horas necesarias para cada tarea, junto con el tiempo real dedicado a cada tarea, y una última columna de comparación entre ambas, obteniendo la desviación.

Como puede comprobarse en la Tabla 2.1, se puede observar que la mayor desviación es en la fase de desarrollo, en especial, el desarrollo backend, el cual se estiman 10 horas más de las usadas finalmente. El resto de las tareas tienen una estimación de tiempo muy similar a el tiempo usado finalmente, con varias estimaciones acertadas. Podemos ver como

Tarea	Tiempo estimado	Tiempo real	Desviación
Elección del proyecto	5h	2h	-3h
Investigación tecnologías	8h	9h	+1h
Entorno de desarrollo	6h	4h	-2h
Aprendizaje Tecnologías	20h	22h	+2h
Elicitación de requisitos	12h	15h	+3h
Modelo de Datos	5h	5h	0h
Análisis y diseño	18h	17h	-1h
Diseño Mockups	15h	15h	0h
Implementación Back-End	60h	50h	-10h
Implementación Front-End	100h	106h	+6h
Pruebas	20h	24h	+4h
Cambios en el código	15h	16h	+1h
Memoria del proyecto	50h	44h	-6h
Total	334h	329h	-5h

Tabla 2.1: Estimación temporal del proyecto

finalmente de las 334 horas estimadas, se han usado 329, por lo que se han consumido 5 horas menos de las estimadas al inicio.

2.2– Diagrama de Gantt

En el diagrama de Gannt, generado mediante la herramienta ClickUp [4] muestra la planificación y desarrollo que se ha llevado a cabo a lo largo del proyecto. Podemos verlo en la siguiente figura 2.1, que ha sido definido en 5 fases o sprints:

- Planificación y estudio, Figura 2.2.
- Análisis y Diseño, Figura 2.3
- Desarrollo y Control, Figura 2.4
- Finalización, Figura 2.5

Para facilitar su visualización e indicar las fechas exactas de comienzo y finalización de cada tarea, se añaden imágenes con cada momento del desarrollo de manera más detallada, con un rango de tiempo menor.

Podemos observar en la figura 2.2 como desde principios de Octubre, cuando contacto por primera vez con el tutor tras pensar en el proyecto

que quería realizar, ocurren también la investigación de tecnologías y el aprendizaje de estas, como pueden ser Solidity, React y web3.

En la figura 2.3 es observable que desde finales de enero, comienza el análisis, con la elicitación de funcionalidades y objetivos, y más tarde los requisitos, con el modelo de datos y los mockups de la aplicación.

La figura 2.2 representa el periodo de implementación de la aplicación, empezando en Mayo, tanto de back-end como de front-end, y continúa hasta finales de Junio, acabando con tests de la aplicación.

Por último en la figura 2.5 se puede observar como se dedica el tiempo a la memoria del proyecto y a cambios menores en la aplicación.

Hay varios momentos en el diagrama que varias tareas se realizan de manera concurrente, debido a que se pueden trabajar en paralelo, o que se complementan, como puede ser el caso del desarrollo back-end y front-end, o el análisis de requisitos y el modelo de datos.

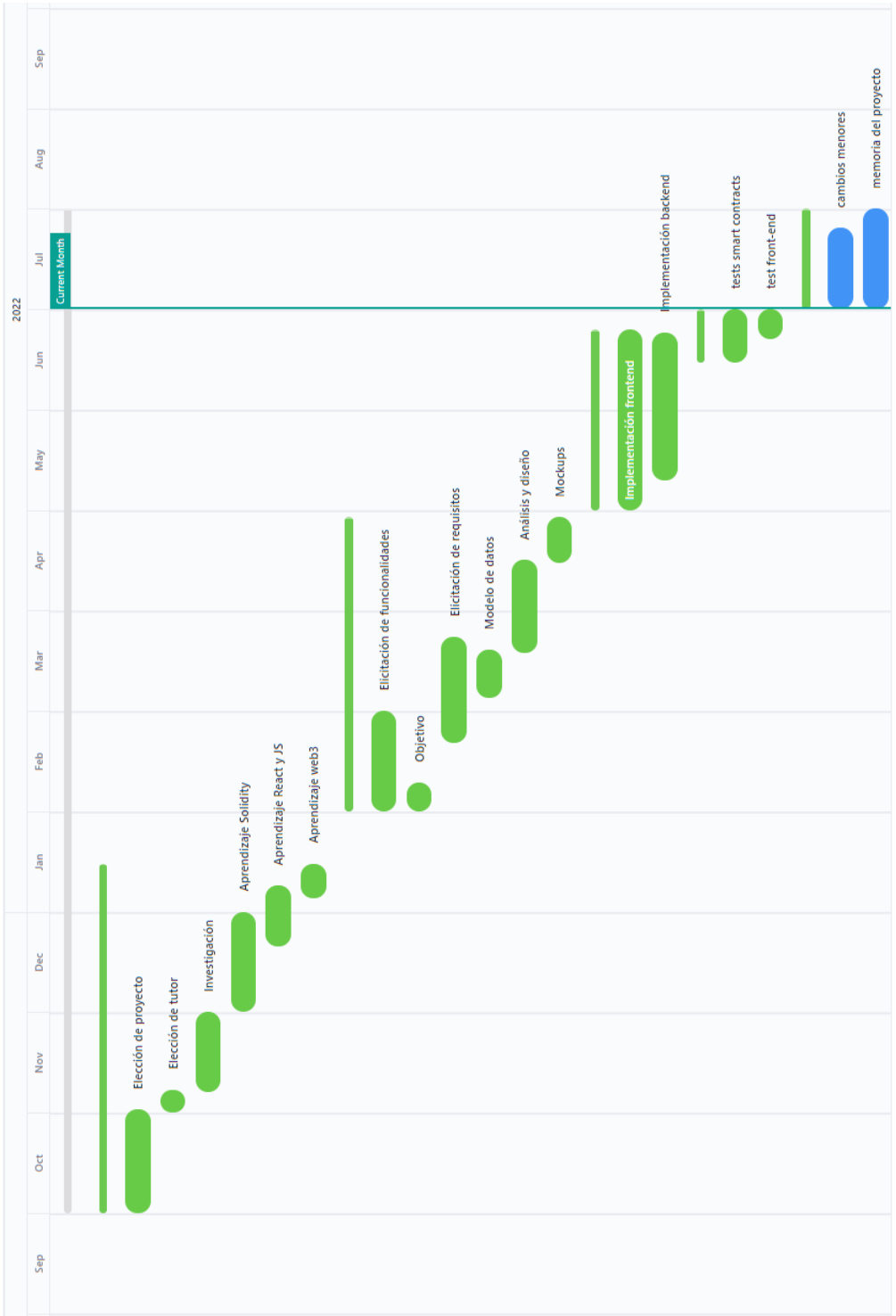

Figura 2.1: Diagrama de Gantt completo.

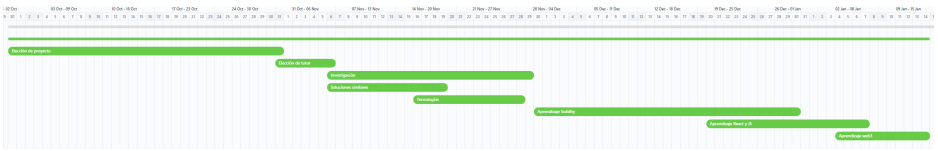

Figura 2.2: Vista detallada planificación del proyecto.

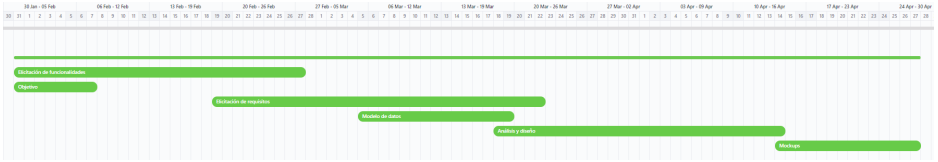

Figura 2.3: Vista detallada análisis y diseño.

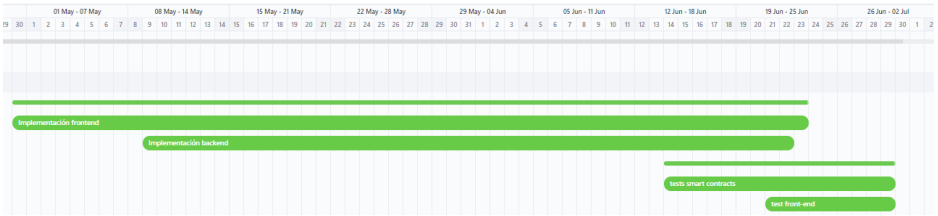

Figura 2.4: Vista detallada desarrollo y control.

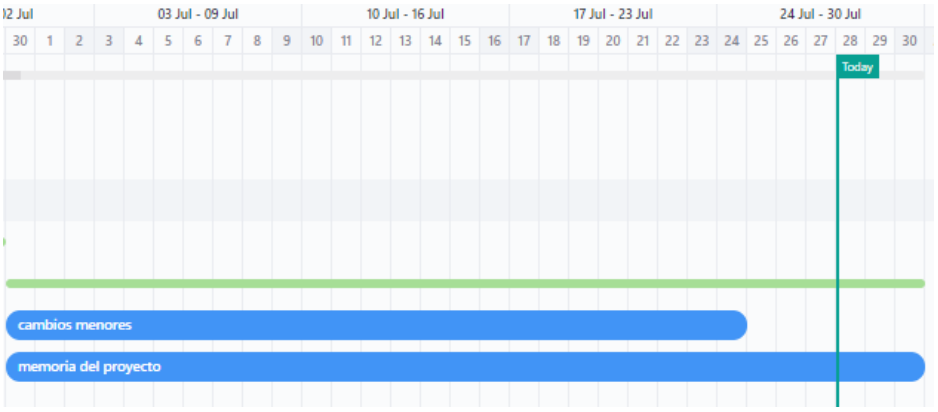

Figura 2.5: Vista detallada finalización.

2.3– Análisis de Costes

En esta sección se desglosará el coste total del desarrollo proyecto, dividido en costes directos e indirectos. Además, se comparará costes estimados con los costes finales, mediante los valores recogidos en la Tabla 2.1.

2.3.1. Costes directos

Gracias a la herramienta GlassDoor [13] se ha realizado una media entre los sueldos de desarrollador Blockchain, React y de Jefe de proyecto, sin contar bonificaciones, siempre en bruto, puesto que dependiendo de la situación del trabajador el salario neto cambiaría para un mismo puesto.

Puesto de Trabajo	**Salario Medio anual**	**Salario Mensual (12 pagas)**
Blockchain Developer	28867,00€	2405,58€
React Developer	29894,00€	2491,16€
Jefe de proyecto	48971,00€	4080,91€
Media	35910,66€	2992,55€

Tabla 2.2: Salarios de las posiciones en nuestro proyecto

Usando la media obtenida en la Tabla 2.2, podemos calcular el salario bruto del desarrollador del proyecto, que son 30 € por hora, contando con que es un desarrollador sin experiencia previa. La Tabla 2.3 expone los costes por tarea del proyecto.

Como se observa, con respecto al tiempo estimado de 334 horas, el coste total del salario se ve reducido en 150 €, lo que supone un decremento menor del 5 % en el valor estimado para el proyecto, y es un valor asumible respecto al tamaño del proyecto.

Tarea	**Horas trabajadas**	**Sueldo**
Elección del proyecto	2h	60€
Investigación tecnologías	9h	270€
Entorno de desarrollo	4h	120€
Aprendizaje tecnologías	22h	660€
Elicitación de requisitos	15h	450€
Modelo de datos	5h	150€
Análisis y diseño	17h	510€
Diseño Mockups	15h	450€
Implementación Back-End	50h	1500€
Implementación Front-End	106h	3180€
Pruebas	24h	720€
Cambios en el código	16h	480€
Memoria del proyecto	44h	1320€
Total	329h	9870€

Tabla 2.3: Coste por tarea

2.3.2. Costes indirectos

La amortización de las herramientas usadas en el desarrollo del proyecto tanto hardware como software componen esta sección. Los dispositivos hardware se calcula que serán amortizados en 3 años, es decir, a un 33 % por año. A continuación, se puede observar los elementos hardware usados, y su precio, junto al coste mensual en la Tabla 2.4.

- Portátil MSI P65 Creator 1000,00€
- Monitor LG ultrawide 140,00€
- Teclado Razer Blackwidow 100,00€
- Auriculares Razer Nari Essential 80,00€
- Ratón Microsoft Surface 80,00€

El desarrollo del proyecto al completo ha sido de 9 meses, por lo tanto, el coste total será el mensual multiplicado por los meses de uso, en este caso, 9.

Hardware	**Coste mensual**	**Total**
Portátil MSI P65 Creator	27,78€	250€
Monitor LG ultrawide	3,89€	35€
Teclado Razer Blackwidow	2,78€	25€
Auriculares Razer Nari Essential	2,23€	20€
Ratón Microsoft Surface	2,23€	20€

Tabla 2.4: Costes de los dispositivos hardware utilizados en el proyecto

El software utilizado en el proyecto no ha supuesto ningún incremento en el coste puesto que se ha trabajado con licencias gratuitas o de estudiante, pero para la formación en el lenguaje de Solidity se han usado cursos gratuitos y de pago. Todo se puede ver en la Tabla 2.5.

Software	**Coste**
GitHub Pro (Estudiante)	0,00€
GitKraken Client Pro (Estudiante)	0,00€
Editor de texto LaTeXOverleaf	0,00€
Visual Studio Code 1.67.2	0,00€
Remix IDE	0,00€
Truffle v5.5.4	0,00€
Ganache v2.6.0-beta.3	0,00€
Figma Starter version	0,00€
Clickup Free Forever Plan	0,00€
Diagrams.net	0,00€
Curso Udemy Solidity	19,99 €
Curso Cryptozombies Solidity	0,00 €

Tabla 2.5: Coste del Software y formación utilizado en el proyecto

2.3.3. Costes totales

Tras desglosar tanto costes indirectos como directos del proyecto, se ha legado a la conclusión que el desarrollo tendría un coste de **10.239,99 €** , que contempla solo la etapa de desarrollo, sin referenciar el despliegue de la solución.

Puesto que hay que firmar transacciones continuamente, se debería buscar una blockchain de segunda capa o con un protocolo donde las transacciones sean de bajo coste, compatibles con EVM, que ha sido el entorno donde se ha desarrollado la aplicación.

CAPÍTULO 3

Elicitación del problema

En este capítulo expondremos la elicitación de requisitos del sistema. Se seguirán las directrices del estándar IEEE 830-1998 [14]. Se dividirán en dos secciones, la primera contemplará alcance y actores del sistema recogida en la Sección 3.1, mientras que la segunda estará formada por los requisitos específicos del sistema en la Sección 3.2 que contendrá los requisitos funcionales 3.2.1, no funcionales 3.2.2, de información 3.2.3 y reglas o requisitos de negocio 3.2.4.

3.1– Introducción

3.1.1. Alcance

El objetivo del sistema es implementar una plataforma que permita el envío y gestión de documentos a través de la tecnología blockchain beneficiando al sistema de algunas de sus propiedades.

3.1.2. Actores del sistema

Se definen los actores que podrán aparecer en el sistema propuesto, para más adelante darle uso en la Sección 3.2.1 e informar de las funciones y requisitos de cada actor.

Actor 1: Empresa	
Descripción	Este actor representa a la empresa registrada en el sistema con los datos necesarios mediante una transacción en la blockchain.

Tabla 3.1: Empresa

Actor 2: Usuario	
Descripción	Este actor representa a una persona que se ha registrado en el sistema con los datos necesarios mediante una transacción en la blockchain. Un usuario podrá tener diferentes niveles: • Nivel 0: el usuario se ha registrado solo con la cartera. • Nivel 1: el usuario se ha registrado con cartera, nombre y apellidos. • Nivel 2: el usuario se ha registrado con todos los datos.

Tabla 3.2: Usuario

Actor 3: Usuario no autenticado	
Descripción	Este actor representa a un usuario o empresa que aún no ha sido registrado en el sistema.

Tabla 3.3: Usuario no autenticado

3.2– Requisitos Específicos

3.2.1. Requisitos funcionales

Los requisitos funcionales son pequeñas descripciones que detallan las funcionalidades y comportamiento del sistema respecto a los diferentes roles que han sido definidos previamente en la Sección 3.1.2.

RF-01	**Página de Landing**
Actor	Usuario no autenticado.
Dependencias	Ninguna.
Descripción	Deberá ser capaz de ver la página de Landing, con posibilidad de interactuar con ella sin ningún tipo de restricción.

Tabla 3.4: Página de Landing.

RF-02	**Página de Inicio**
Actor	Usuario no autenticado.
Dependencias	Ninguna.
Descripción	Deberá ser capaz de ver la página de Inicio, con posibilidad de conectar su billetera y de registrarse como usuario.

Tabla 3.5: Página de Inicio.

RF-03	**Página de Inicio para empresas**
Actor	Usuario no autenticado.
Dependencias	Ninguna.
Descripción	Deberá ser capaz de ver la página de Inicio, con posibilidad de conectar su billetera y de registrarse como una empresa.

Tabla 3.6: Página de Inicio para empresas.

RF-04	**Registro de Usuario**
Actor	Usuario no autenticado.
Dependencias	RI-01, RI-02, RI-03.
Descripción	Deberá poder ver el formulario de registro de usuario para introducir los datos mínimos requeridos y acabar firmando la transacción para que el registro sea exitoso.

Tabla 3.7: Registro de Usuario.

RF-05	**Registro de empresa**
Actor	Usuario no autenticado.
Dependencias	RI-04.
Descripción	Deberá poder ver el formulario de registro de empresa para introducir los datos mínimos requeridos y acabar firmando la transacción para que el registro sea exitoso.

Tabla 3.8: Registro de usuario en organización.

RF-06	**Listar empresas disponibles**
Actor	Usuario.
Dependencias	Ninguna.
Descripción	Deberá poder ver una lista con todas las empresas que se han registrado en la aplicación.

Tabla 3.9: Listar empresas disponibles.

RF-07	**Listar documentos enviados**
Actor	Usuario.
Dependencias	RN-01, RI-05.
Descripción	Deberá poder ver una lista con todos los documentos que ha mandado a las diferentes empresas.

Tabla 3.10: Listar empresas disponibles.

RF-08	**Listar documentos recibidos**
Actor	Empresa.
Dependencias	RN-01, RI-05.
Descripción	Deberá poder ver una lista con todos los documentos que ha recibido por parte de los diferentes usuarios.

Tabla 3.11: Listar empresas disponibles.

RF-09	**Listar información personal**
Actor	Usuario, Empresa.
Dependencias	Ninguna.
Descripción	Deberá poder ver la información personal o perfil que ha proporcionado en la aplicación.

Tabla 3.12: Listar información personal.

RF-10	**Editar información personal**
Actor	Usuario, Empresa.
Dependencias	RI-01, RI-02, RI-03, RI-04.
Descripción	Deberá poder editar la información personal o perfil que ha proporcionado en la aplicación.

Tabla 3.13: Editar información personal.

RF-11	**Añadir o editar foto de perfil**
Actor	Usuario, Empresa.
Dependencias	RI-01, RI-02, RI-03, RI-04.
Descripción	Deberá poder añadir o editar la imagen de perfil que ha proporcionado en la aplicación.

Tabla 3.14: Añadir o editar foto de perfil.

RF-12	**Acceder a la información del documento**
Actor	Usuario, Empresa.
Dependencias	RN-04.
Descripción	Deberá poder permitir el acceso a los documentos enviados o recibidos, dependiendo del actor, para ver información detallada sobre este y el estado en el que se encuentra.

Tabla 3.15: Acceder a la información del documento

RF-13	**Editar el estado del documento**
Actor	Empresa.
Dependencias	RN-04, RN-02, RI-06.
Descripción	Deberá poder permitir la edición del estado del documento indicando así el punto en el que se encuentra.

Tabla 3.16: Editar el estado del documento

RF-14	**Ver y descargar el documento como Usuario**
Actor	Usuario.
Dependencias	RN-04.
Descripción	Deberá poder permitir ver y descargar el documento que se ha enviado y que se encuentra en el sistema de almacenamiento distribuido sin ningún tipo de restricción.

Tabla 3.17: Ver y descargar el documento como Usuario

RF-15	**Ver y descargar el documento como Empresa**
Actor	Empresa.
Dependencias	RN-04, RI-06.
Descripción	Deberá poder permitir ver y descargar el documento que se ha enviado y que se encuentra en el sistema de almacenamiento distribuido firmando previamente una transacción y actualizando el estado de este.

Tabla 3.18: Ver y descargar el documento como Empresa

RF-16	**Ver información detallada de la empresa**
Actor	Usuario.
Dependencias	RI-01, RI-02, RI-03.
Descripción	Deberá poder permitir ver la información detallada que ha proporcionado una empresa para poder comprobar si es el destinatario correcto.

Tabla 3.19: Ver información detallada de la empresa

RF-17	**Enviar documento**
Actor	Usuario.
Dependencias	RN-01.
Descripción	Deberá poder enviar mensajes con documento adjunto a la empresa seleccionada, completando los campos necesarios.

Tabla 3.20: Enviar documento

3.2.2. Requisitos no funcionales

A continuación, expondremos los requisitos no funcionales o atributos de calidad, los cuales dictan el funcionamiento del sistema, junto con sus restricciones y condiciones a tener en cuenta en las Tablas 3.21, 3.22, 3.23, 3.24, 3.25 y 3.26.

RNF-01	**Velocidad de respuesta**
Descripción	Las transacciones deberán aparecer como máximo en 60000 milisegundos para que se considere válida.

Tabla 3.21: Velocidad de respuesta.

RNF-02	**Compatibilidad navegadores y dispositivos**
Descripción	Deberá ser posible usar la solución con cualquier dispositivo siempre y cuando el navegador cuente con un cliente ligero de conexión a billtera, por ejemplo, Metamask [16].

Tabla 3.22: Compatibilidad con navegadores y dispositivos.

RNF-03	**Integridad de los datos**
Descripción	Deberá guardar y recuperar los datos almacenados en los sistema distribuidos de manera íntegra.

Tabla 3.23: Integridad de los datos.

RNF-04	**Foto de perfil por defecto**
Descripción	Deberá proveer de una imagen de perfil por defecto cuando el usuario no haya seleccionado ninguna.

Tabla 3.24: Foto de perfil por defecto.

RNF-05	**Disponibilidad de los datos**
Descripción	Deberá ofrecer el acceso a la información independientemente del lugar y momento, siempre y cuando los sistemas distribuidos estén desplegados y disponibles.

Tabla 3.25: Disponibilidad de los datos.

RNF-07	**Interfaz de usuario**
Descripción	Deberá contar con una interfaz de usuario sencilla e intuitiva, que provea de las funcionalidades de la solución.

Tabla 3.26: Interfaz de usuario.

3.2.3. Requisitos de información

La información que se almacena en los sistemas distribuidos se expone en los requisitos de información, datos que la solución almacenará para cumplir los requisitos expuestos anteriormente en la Sección 3.2.1.

RI-01	**Usuario Nivel 0**
Descripción	El sistema debe almacenar los siguientes datos de un usuario para considerarse de Nivel 0.
Datos almacenados	• Dirección de billetera. • Hash de la foto de perfil.

Tabla 3.27: Datos del usuario nivel 0.

RI-02	**Usuario Nivel 1**
Descripción	El sistema debe almacenar los siguientes datos de un usuario para considerarse de Nivel 1.
Datos almacenados	• Dirección de billetera. • Nombre. • Apellidos. • Hash de la foto de perfil.

Tabla 3.28: Datos del usuario nivel 1.

RI-03	**Usuario Nivel 2**
Descripción	El sistema debe almacenar los siguientes datos de un usuario para considerarse de Nivel 2.
Datos almacenados	• Dirección de billetera. • Nombre. • Apellidos. • Número de identificación personal (DNI). • Teléfono. • Email. • País. • Región. • Código postal. • Ciudad. • Calle. • Hash de la foto de perfil.

Tabla 3.29: Datos del usuario nivel 2.

RI-04	**Empresa**
Descripción	El sistema debe permitir almacenar los datos de una empresa.
Datos almacenados	• Dirección de billetera. • Nombre de la empresa. • Razón Social. • Número de identificación fiscal. • Teléfono. • Email. • País. • Región. • Código postal. • Ciudad. • Calle. • Nivel mínimo requerido por un usuario para interactuar con la empresa. • Hash de la foto de perfil.

Tabla 3.30: Datos de la empresa.

RI-05	**Mensaje**
Descripción	El sistema debe permitir almacenar siguientes datos en el mensaje que se envía a una empresa.
Datos almacenados	• id del mensaje. • Asunto. • Cuerpo del mensaje. • Usuario que lo envía. • Empresa a la que va dirigido. • Hash del documento. • Estado del documento.

Tabla 3.31: Datos almacenados de un mensaje.

RI-06	**Estados de un mensaje**
Descripción	El sistema debe permitir almacenar los siguientes estados posibles de un mensaje.
Datos almacenados	• Enviado: cuando el mensaje llega al destinatario. • Descargado: cuando el destinatario descarga el documento. • Confirmado: tras la descarga, cuando se da el visto bueno al documento. • Rechazado: tras la descarga, cuando no se le da el visto bueno al documento.

Tabla 3.32: Datos del estado de un mensaje.

3.2.4. Reglas de negocio

Para acotar de una manera más precisa algunas funcionalidades de la solución, se exponen restricciones en estas funcionalidades en específico.

RN-01	**Envío de documentos**
Actor	Usuario.
Descripción	Para poder realizar un envío de documentos, el usuario debe igualar o superar el nivel mínimo de datos requerido por la organización.

Tabla 3.33: Restricción de envío de documentos.

RN-02	**Estado del documento**
Actor	Empresa.
Descripción	Un documento no se podrá marcar como confirmado o rechazado si ya tiene uno de esos estados.

Tabla 3.34: Restricciones en el estado de documentos.

RN-03	**Cartera única**
Actor	Usuario, Empresa.
Descripción	Si ya hay una dirección registrada como usuario o empresa, no podrá obtener el otro rol y deberá usar una dirección de cartera distinta.

Tabla 3.35: Restriccion de cartera.

RN-04	**Permiso de lectura**
Actor	Usuario, Empresa.
Descripción	Sólo será posible leer el contenido de un mensaje si eres el remitente o destinatario.

Tabla 3.36: Permiso de lectura.

CAPÍTULO 4

Análisis y diseño

En este capítulo, expondremos de forma detallada la fase de ánalisis y diseño que se ha seguido para la solución. Gracias a la cual, conseguimos una visión mucho más acertada de la solución final, siguiendo los requisitos que se expusieron en el capítulo anterior.

La primera Sección, 4.1 mostrará los diagramas de secuencia producidos al usar la aplicación descentralizada. Explicaremos también en la Sección 4.2 que muestra al completo los casos de uso junto con las tablas detalladas de cada uno de ellos. Un pequeño modelo de datos describiendo atributos y relaciones de nuestra solución en la Sección 4.3. Un diagrama que muestra la arquitectura que usa nuestra solución, y, por último, los mockups realizados previos a la implementación en la Sección 4.5.

4.1– Descripción de diagramas de secuencia

Para realizar la conexión entre la blockchain elegida y nuestra solución, debemos usar un proveedor que haga de intermediario para interactuar con la dApp. En este caso, el proveedor será la billetera, que nos permite firmar transacciones en el momento que queramos interactuar con alguna función del contrato inteligente que ha sido programado y desplegado en nuestra red de pruebas.

Se ha creado un diagrama de secuencia para los casos de uso en los que se interactúa con la blockchain, de manera genérica, puesto que los cambios son los datos que se envían, los actores y las funciones a las que se llaman, pero para todos estos casos, la secuencia es la misma, que se puede ver en la Figura 4.1. El primer paso es que el usuario rellene los datos pertinentes, de un formulario. Al enviar estos datos, la billetera contacta con la blockchain, la cual recibe y comprueba los datos, y cuando

los acepta, manda firmar al usuario la transacción. Una vez el usuario la firme, se envían todos los datos a la blockchain y esta confirma una vez se hayan registrado los datos dentro de esta.

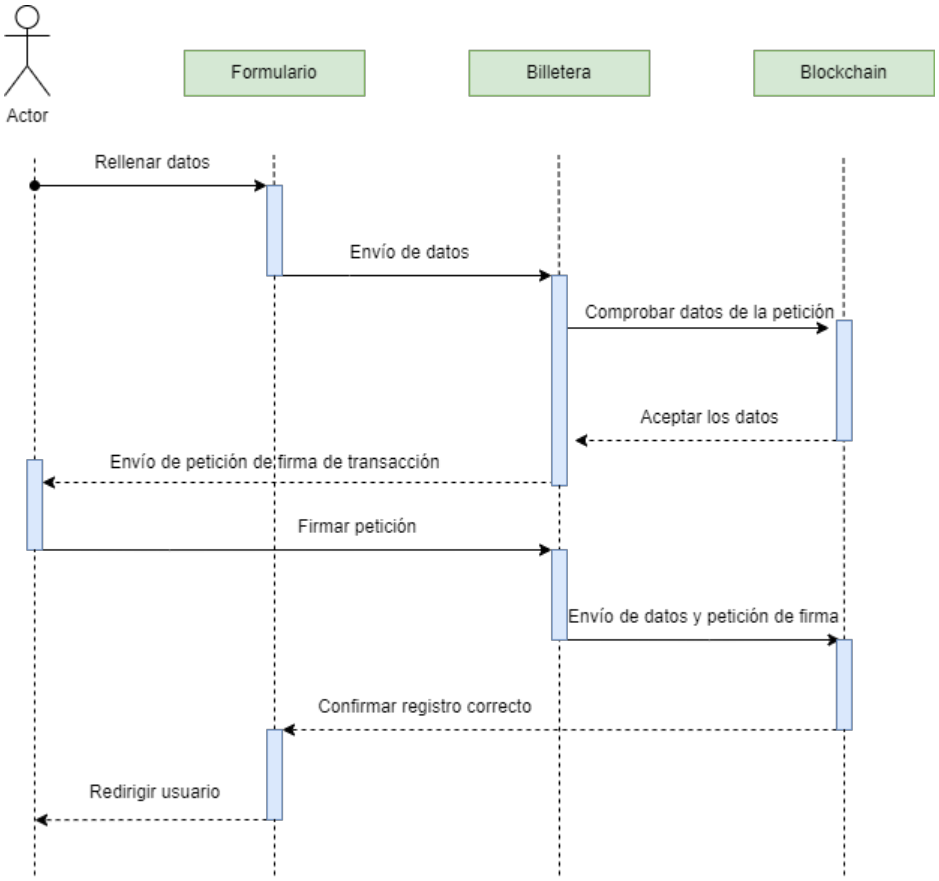

Figura 4.1: Diagrama de secuencia para comunicación con Blockchain.

También se ha creado otro diagrama, visible en la Figura 4.2, para explicar la conexión con el sistema distribuido de almacenamiento, y la blockchain al mismo tiempo, puesto que se añaden pasos para obtener el hash de nuestro archivo. En este ejmplo lo haremos con la imagen de perfil, el usuario sube la foto que quiere tener, y el sistema la codifica, para después enviarla al sistema de archivo distribuido, el cual nos devuelve un hash donde está localizada la imagen. Este hash será enviado a la billetera, la cual hará la petición a la blockchain para que compruebe los datos, y pedirá al usuario que firme la transacción. Una vez que la firma se hace, los datos son escritos en la blockchain y se confirma la transacción. Para que la aplicación, realice una petición de datos por la cual podremos ver

nuestra imagen de perfil correctamente actualizada.

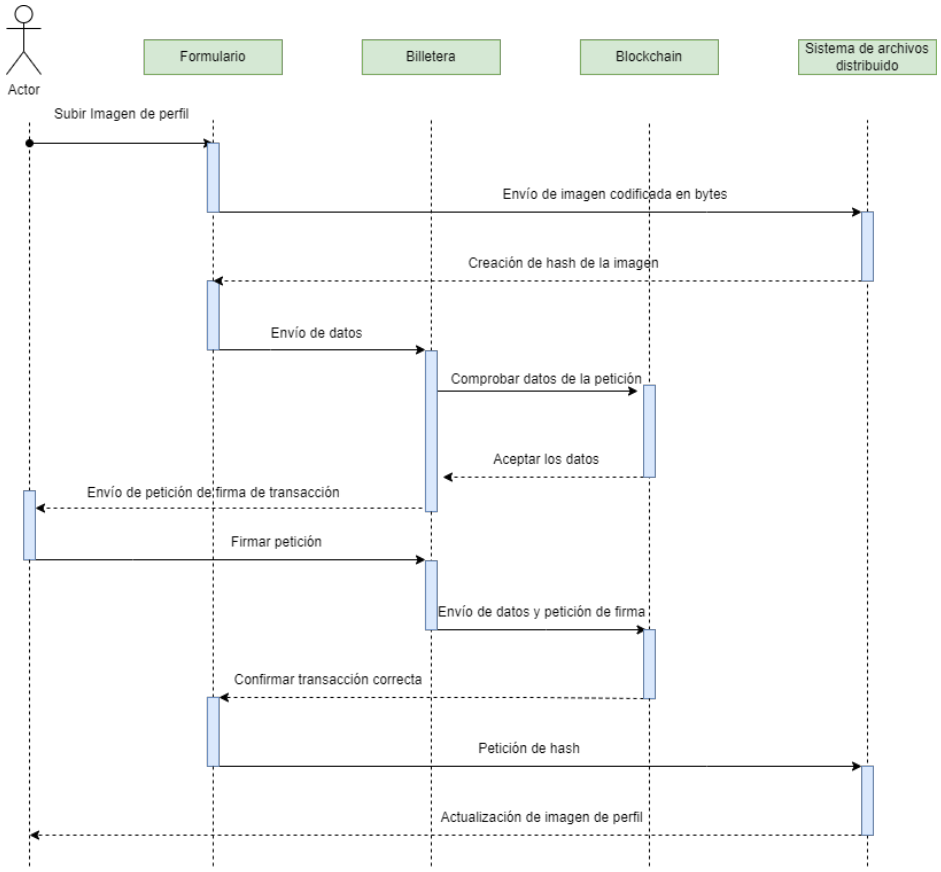

Figura 4.2: Diagrama de secuencia para comunicación con Sistema de archivos distribuidos y Blockchain.

4.2– Modelo de casos de uso

En esta sección, mostraremos un diagrama con los casos de uso conectados al actor que puede realizarlos.

Cada bloque hace referencia a un conjunto de funcionalidades sobre un apartado en concreto, y los actores, apuntan al bloque completo, o en el caso en el que no puedan realizar todas las funcionalidades de ese bloque, solo apuntan a las que pueden usar.

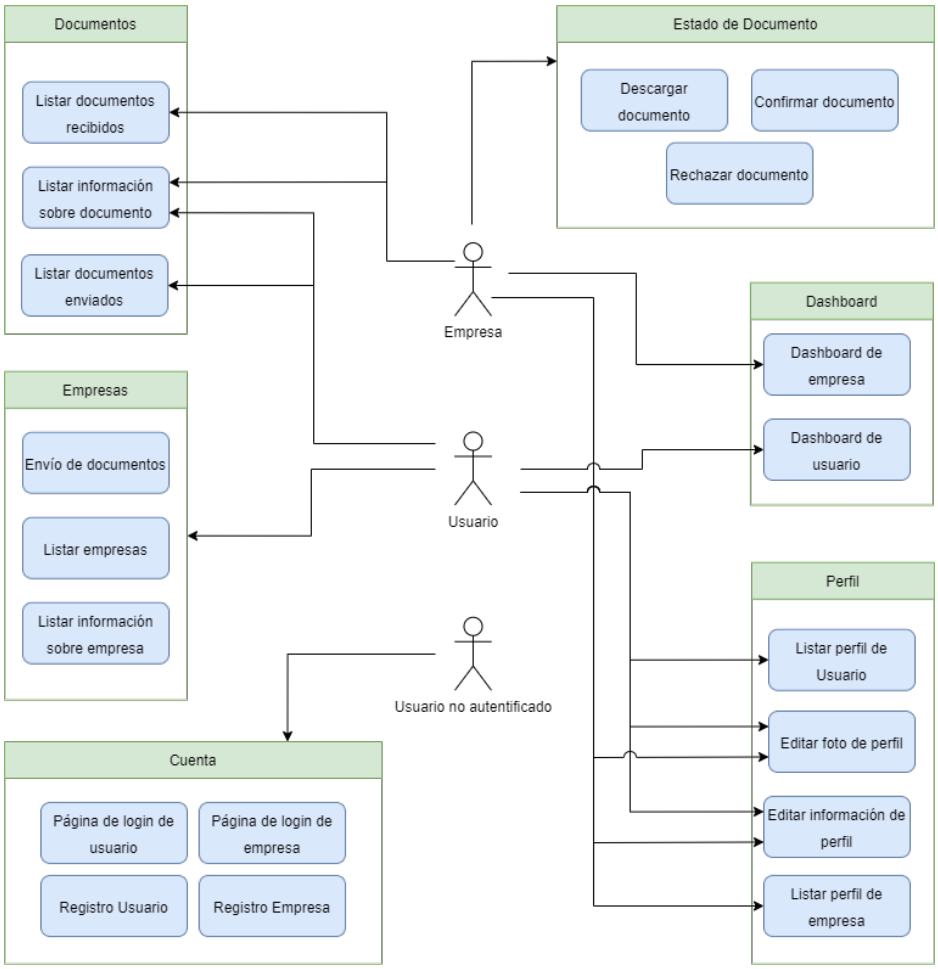

Figura 4.3: Diagrama casos de uso.

Procedemos a explicar de manera detallada cada caso de uso indicado en el diagrama anterior, en forma de tabla, con los pasos a realizar.

CU-00	**Ver página de landing**	
Dependencias	RF-01	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario no autenticado quiera ver la página de landing.	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario no autenticado accede a la página de landing mediante enlace.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.1: Ver página de landing.

CU-01	**Ver página de login de usuario**	
Dependencias	RF-02	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario no autenticado quiera ver la página de login de usuario.	
Precondición	El usuario no debe estar registrado como usuario y debe estar en la página de landing.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario no autenticado accede a la aplicación mediante el enlace "Go to app".
	2	El sistema muestra la página de login de usuario.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.2: Ver página de login de usuario.

CU-02	**Ver página de login de empresa**	
Dependencias	RF-03	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario no autenticado quiera ver la página de login de empresas.	
Precondición	El usuario no debe estar registrado como empresa y debe estar en la página de landing.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario no autenticado accede a la aplicación mediante el enlace "Go to app".
	2	El sistema muestra la página de login de usuario.
	3	El usuario accede a la sección " login de empresa".
	4	La aplicación muestra la página de login de empresa.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.3: Ver página de login de empresa.

CU-03	**Registro de usuario**	
Dependencias	RF-02, RF-04	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario no autenticado quiera registrarse como un usuario.	
Precondición	El usuario no debe estar registrado como usuario y debe estar en la página de landing.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario no autenticado accede a la aplicación mediante el enlace "Go to app".
	2	El sistema muestra la página de login de usuario.
	3	El usuario pulsa sobre "registro".
	4	La aplicación muestra el formulario de registro.
	5	El usuario rellena los campos pertinentes del formulario.
	6	El sistema pide firmar la transacción con la billetera.
	7	El usuario firma la transacción.
	8	El sistema inicia sesión y envía al usuario a "Dashboard".
Postcondición	Se almacenan los datos del nuevo usuario en la blockchain.	
Excepciones	Si ya existe la cartera, mostrará un mensaje de error.	

Tabla 4.4: Registro de usuario.

CU-04	**Registro de empresa**	
Dependencias	RF-03, RF-05	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario no autenticado quiera registrarse como una empresa.	
Precondición	El usuario no debe estar registrado como empresa y debe estar en la página de landing.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario no autenticado accede a la aplicación mediante el enlace "Go to app".
	2	El sistema muestra la página de login de usuario.
	3	El usuario accede a la sección "login de empresa".
	4	La aplicación muestra la página de login de empresa.
	5	El usuario pulsa sobre "registro".
	6	El usuario rellena los campos pertinentes del formulario.
	7	El sistema pide firmar dos transacciones con la billetera.
	8	El usuario firma la transacción.
	9	El sistema inicia sesión y envía al usuario a "Dashboard".
Postcondición	Se almacenan los datos de la nueva empresa en la blockchain.	
Excepciones	Si ya existe la cartera, mostrará un mensaje de error.	

Tabla 4.5: Registro de empresa.

CU-05	**Ver perfil de usuario**	
Dependencias	RF-09	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver su perfil.	
Precondición	El usuario debe haber iniciado sesión como usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Perfil".
	2	El sistema muestra la página de perfil del usuario.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.6: Ver perfil de usuario.

CU-06	**Ver perfil de empresa**	
Dependencias	RF-09	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver su perfil de empresa.	
Precondición	El usuario debe haber iniciado sesión como empresa y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Perfil".
	2	El sistema muestra la página de perfil de la empresa.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.7: Ver perfil de empresa.

CU-07	**Editar foto de perfil**	
Dependencias	RF-11	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera editar su foto de perfil.	
Precondición	El usuario debe haber iniciado sesión como empresa o usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Perfil".
	2	El sistema muestra la página de perfil de la empresa.
	3	El usuario selecciona la imagen.
	4	El usuario pulsa sobre "cambiar foto de perfil".
	5	El sistema pide firmar una transacción con la billetera.
	6	El usuario firma la transacción.
	7	Se muestra la página de perfil con la nueva imagen.
Postcondición	La foto de perfil de la cuenta cambia.	
Excepciones	Ninguna.	

Tabla 4.8: Editar foto de perfil.

CU-08	**Editar información de perfil**	
Dependencias	RF-10	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera editar su foto de perfil.	
Precondición	El usuario debe haber iniciado sesión como empresa o usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Perfil".
	2	El sistema muestra la página de perfil de la empresa.
	3	El usuario cambia los datos que desea.
	4	El usuario pulsa sobre enviar los nuevos datos.
	5	El sistema pide firmar una transacción con la billetera.
	6	El usuario firma la transacción.
	7	El sistema actualiza la información y la muestra.
Postcondición	La información de la cuenta cambia.	
Excepciones	Ninguna.	

Tabla 4.9: Editar información de perfil.

CU-09	**Ver dashboard de usuario**	
Dependencias	RF-04	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver su perfil.	
Precondición	El usuario debe haber iniciado sesión como usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Dashboard".
	2	El sistema muestra la dashboard del usuario.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.10: Ver dashboard de usuario.

CU-10	**Ver dashboard de empresa**	
Dependencias	RF-05	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver su perfil.	
Precondición	El usuario debe haber iniciado sesión como empresa y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Dashboard".
	2	El sistema muestra la dashboard de la empresa.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.11: Ver dashboard de empresa.

CU-11	**Ver documentos enviados**	
Dependencias	RF-07	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver los documentos que ha enviado en el pasado.	
Precondición	El usuario debe haber iniciado sesión como usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Documentos".
	2	El sistema muestra todo el listado de documentos que envió.
Postcondición	Ninguna.	
Excepciones	Si no ha enviado documentos, no mostrará ninguno.	

Tabla 4.12: Ver documentos enviados.

CU-12	**Ver documentos recibidos**	
Dependencias	RF-08	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver los documentos que ha recibido.	
Precondición	El usuario debe haber iniciado sesión como empresa y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Documentos".
	2	Se muestra los documentos que han sido recibidos.
Postcondición	Ninguna.	
Excepciones	Si no ha recibido documentos, no mostrará ninguno.	

Tabla 4.13: Ver documentos recibidos.

CU-13	**Ver la información de un documento**	
Dependencias	RF-12	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver la información de un documento.	
Precondición	El usuario debe haber iniciado sesión como empresa o usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Documentos".
	2	Se muestran los documentos.
	3	El usuario pulsa sobre uno de ellos.
	4	El sistema muestra la información detallada sobre el documento.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.14: Ver la información de un documento.

CU-14	**Descargar un documento**	
Dependencias	RF-14, RF-15	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera descargar un documento que ha recibido.	
Precondición	El usuario debe haber iniciado sesión como empresa, tener la billetera conectada y estar en la vista de la información de un documento.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario descarga un documento.
	2	El sistema pide firmar una transacción con la billetera.
	3	El usuario firma la transacción.
	4	El sistema descarga el documento para el usuario.
Postcondición	El usuario puede ver el documento y tenerlo en el dispositivo. El sistema actualiza el estado del documento a "descargado".	
Excepciones	Ninguna.	

Tabla 4.15: Descargar un documento.

CU-15	Confirmar un documento	
Dependencias	RF-12, RF-13	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera confirmar un documento.	
Precondición	El usuario debe haber iniciado sesión como empresa, tener la billetera conectada y estar en la vista de la información de un documento.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario confirma un documento.
	2	El sistema pide firmar una transacción con la billetera.
	3	El usuario firma la transacción.
	4	El sistema confirma el documento.
Postcondición	El sistema actualiza el estado del documento a "confirmado".	
Excepciones	Si el documento ya ha sido confirmado o rechazado previamente, no funcionará.	

Tabla 4.16: Confirmar un documento.

CU-16	Rechazar un documento	
Dependencias	RF-12, RF-13	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera rechazar un documento.	
Precondición	El usuario debe haber iniciado sesión como empresa, tener la billetera conectada y estar en la vista de la información de un documento.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario rechaza un documento.
	2	El sistema pide firmar una transacción con la billetera.
	3	El usuario firma la transacción.
	4	El sistema rechaza el documento.
Postcondición	El sistema actualiza el estado del documento a "rechazado".	
Excepciones	Si el documento ya ha sido confirmado o rechazado previamente, no funcionará.	

Tabla 4.17: Rechazar un documento.

CU-17	**Ver empresas registradas**	
Dependencias	RF-06	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver las empresas registradas.	
Precondición	El usuario debe haber iniciado sesión como usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Empresas".
	2	Se muestra todo el listado de empresas.
Postcondición	Ninguna.	
Excepciones	Si no existen empresas registradas, no mostrará ninguna.	

Tabla 4.18: Ver empresas registradas.

CU-18	**Ver información sobre empresa**	
Dependencias	RF-16	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver información sobre una empresa.	
Precondición	El usuario debe haber iniciado sesión como usuario y tener la billetera conectada.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede a la sección "Empresas".
	2	Se muestra todo el listado de empresas.
	3	El usuario pincha sobre una de las empresas.
	4	Se muestra toda la información de la empresa.
Postcondición	Ninguna.	
Excepciones	Ninguna.	

Tabla 4.19: Ver información sobre empresa.

CU-19	**Enviar documento a empresa**	
Dependencias	RF-17	
Descripción	El sistema se comportará como se expone en el siguiente caso de uso cuando un usuario quiera ver información sobre una empresa.	
Precondición	El usuario debe haber iniciado sesión como usuario,tener la billetera conectada y estar en la vista de información sobre una empresa.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario envia un documento.
	2	El sistema muestra una vista para enviar el documento.
	3	El usuario rellena los campos y selecciona un archivo.
	4	El usuario pulsa sobre enviar.
	5	El sistema pide firmar una transacción con la billetera.
	6	El usuario firma la transacción.
	5	El sistema envía al usuario a la página de inicio.
Postcondición	Habrá un nuevo documento en la pestaña de documentos.	
Excepciones	Ninguna.	

Tabla 4.20: Enviar documento a empresa.

4.3– Modelo de datos

A continuación, se muestra el modelo de datos de nuestra solución, con sus relaciones entre entidades y los atributos de cada una de ellas, como se puede observar en la figura 4.4.

El primer atributo de cada entidad será la clave única para esa clase. Podemos ver como la entidad Documento es la central, y la que relaciona a la empresa con los usuarios, y el documento sólo puede pertenecer a un emisor y a un receptor.

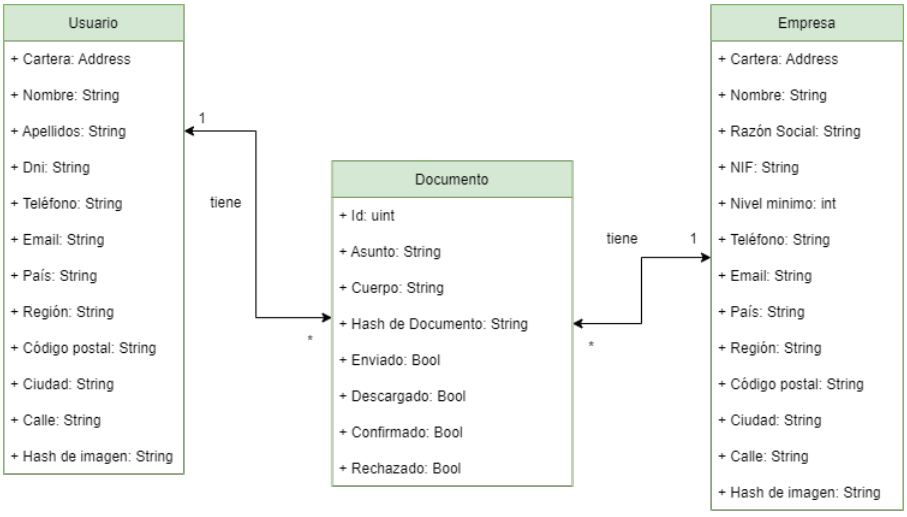

Figura 4.4: Modelo de datos.

4.4– Arquitectura del sistema

Se va a exponer la arquitectura del sistema. Puesto que es una implementación que usa varios sistemas descentralizados, de blockchain y de almacenamiento distribuido, no es el simple modelo de cliente-servidor, puesto que habrá tantos servidores como nodos contenga la blockchain y el sistema distribuido. Podemos ver un diagrama de este esquema en la figura 4.5.

El sistema cuenta con varias partes diferenciadas, front-end, back-end y el sistema de almacenamiento distribuido. El back-end se trata del contrato desplegado en la blockchain. Este contrato, que corre dentro de una máquina virtual propia de la blockchain, cuenta con todas las funcionalidades del sistema y se encarga de interactuar con la blockchain a través de la máquina virtual, para almacenar o acceder a los datos existentes.

El front-end provee al usuario de una interfaz por la cual llamará a las diferentes funcionalidades del sistema, y mostrará los resultados por pantalla. Además de una billetera la cual el usuario usará para firmar transacciones que serán emitidas a la blockchain.

Para la conexión con el back-end, usamos la billetera del front-end, que un proveedor nos ofrece. Además de proveernos de una billetera, hay proveedores que ofrecen sus nodos para almacenar datos en algún sistema de almacenamiento distribuido, que usaremos para guardar los documentos de nuestra solución.

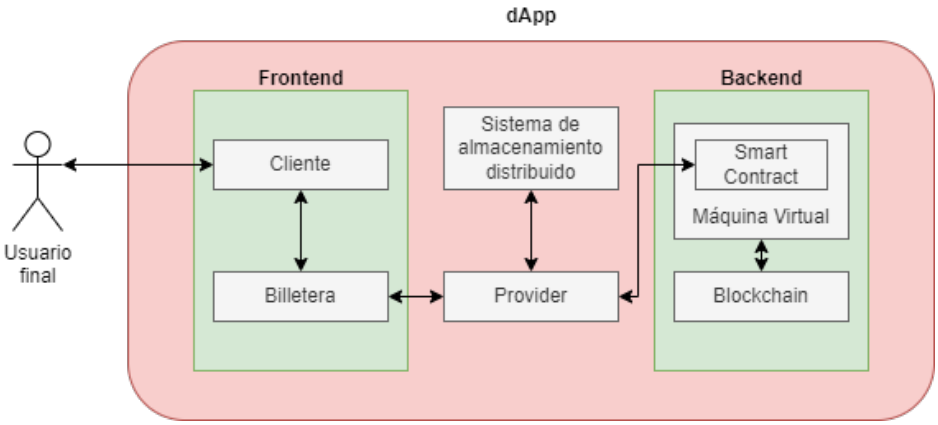

Figura 4.5: Arquitectura del sistema.

4.5– Mockups

Para esta última sección del capítulo, hemos usado la herramienta de figma [8]. Esta herramienta nos permite crear una maqueta de nuestra solución para tener en mente el diseño que seguirá, el cual puede ser fiel o simplemente una idea que sigamos.

A continuación, se muestran algunas de las maquetas hechas para las vistas que después se harán en nuestra solución.

En la figura 4.6, se ideó como debería ser la página de inicio que vería el usuario al entrar, con una explicación sobre la solución y las diferentes secciones de información.

La figura 4.7 sería la página para iniciar sesión según cada actor, la cual, finalmente, difiere de gran manera frente a la vista en la aplicación.

La figura 4.8, representa cómo una empresa registraría sus datos, que es muy similar a la del resto de actores.

La vista principal que un usuario vería una vez iniciada la sesión se ve representada en la figura 4.9.

En la figura 4.10, se representa la sección del perfil de un usuario de nivel 2 en la que están todos sus datos, muy similar a la de una empresa.

En las figuras 4.11 y 4.12 se muestra el proceso que un usuario sigue para entregar un documento a una empresa, previamente visualizando la información.

Por último, en las figuras 4.13 y 4.14 se muestra como un usuario vería los documentos que entregó, y si pincha en uno, la información detallada sobre este.

Figura 4.6: Página de landing.

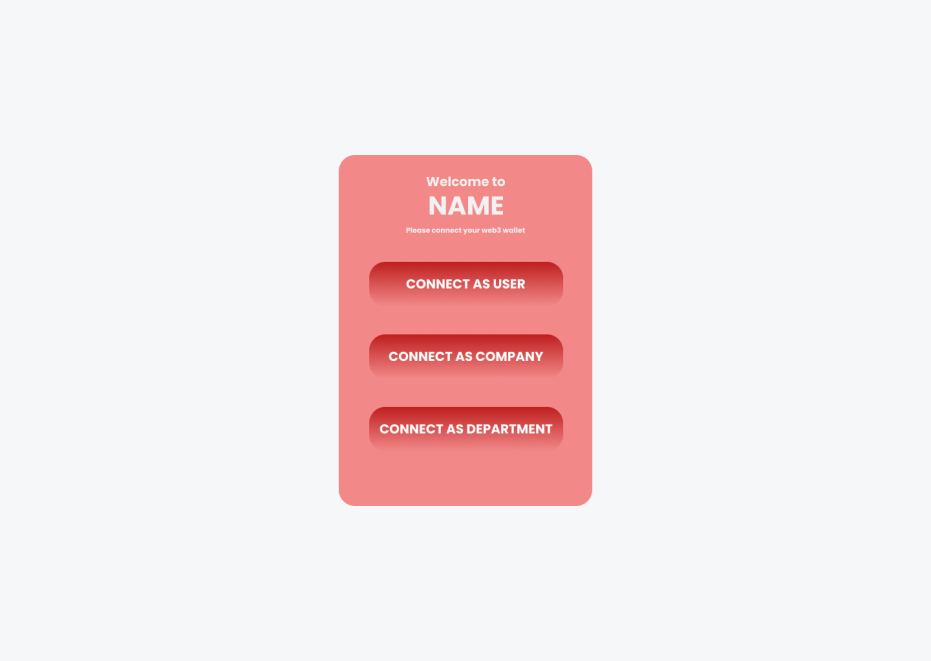

Figura 4.7: Página de inicio de sesión.

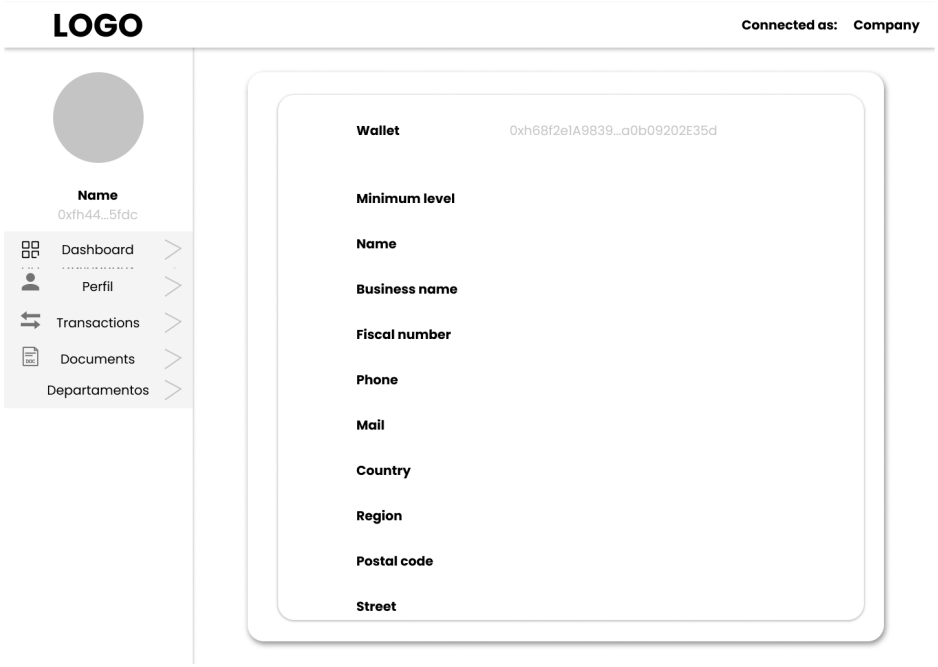

Figura 4.8: Página de registro de empresa.

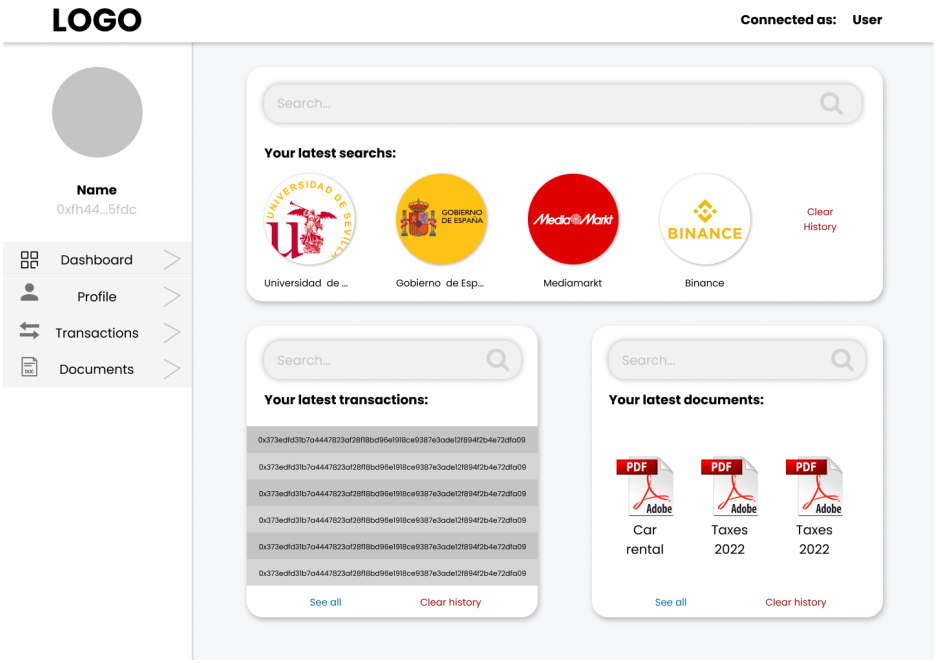

Figura 4.9: Dashboard de usuario.

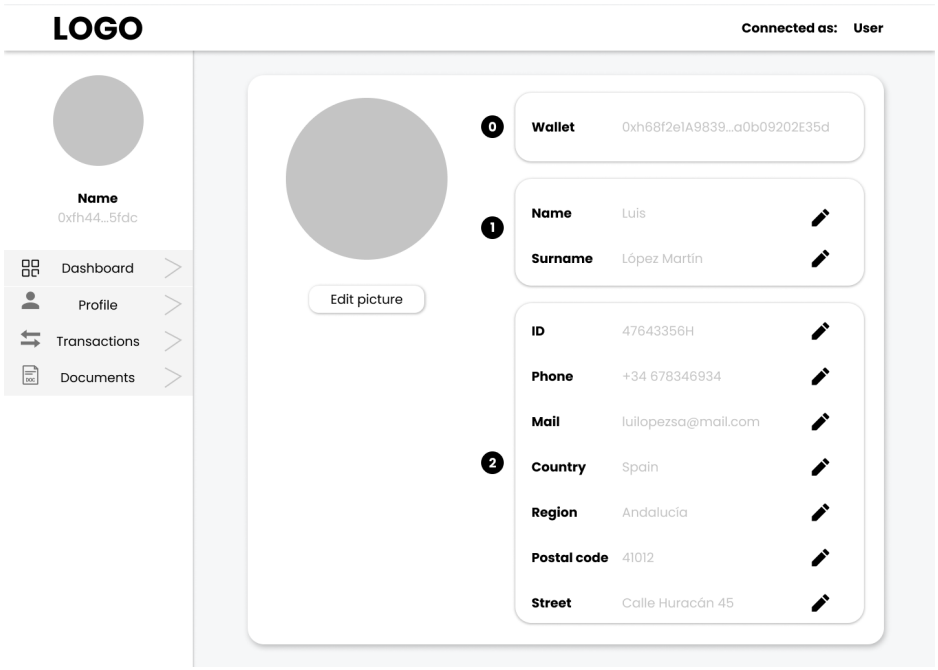

Figura 4.10: Perfil de usuario de nivel 2.

Figura 4.11: Información sobre una empresa.

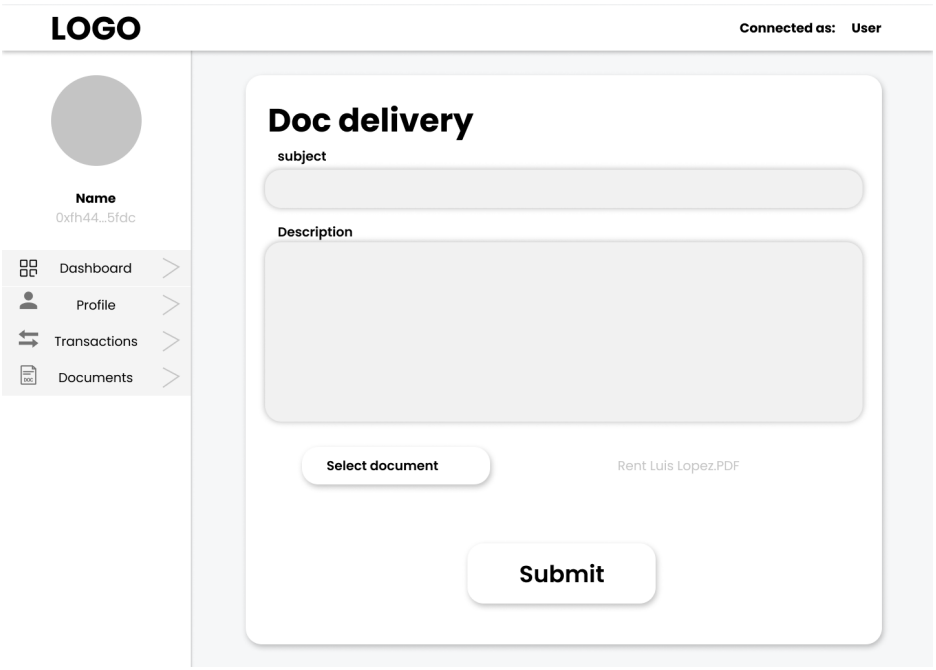

Figura 4.12: Entrega de documentos a empresa.

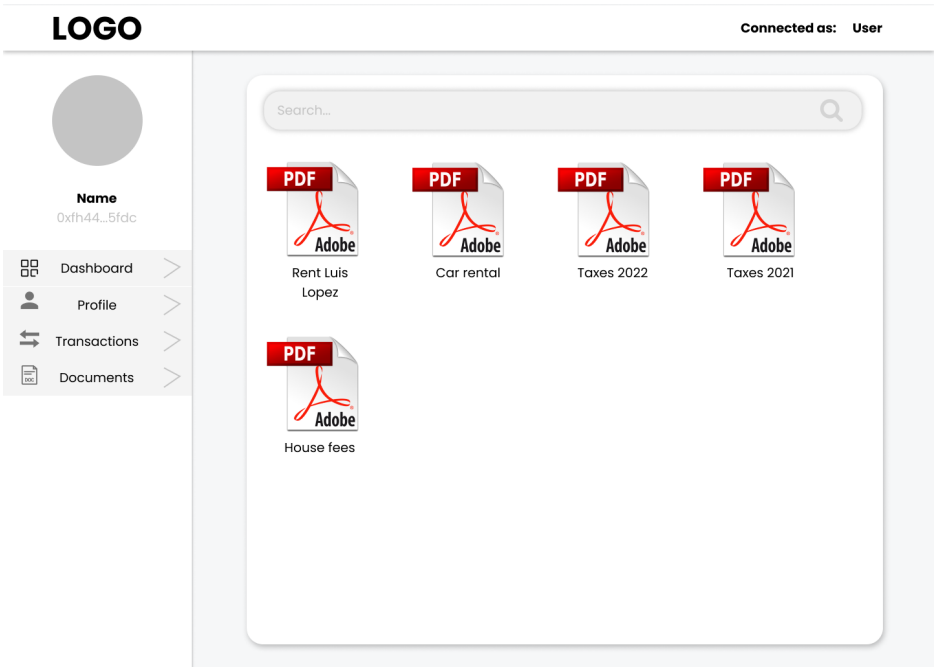

Figura 4.13: Vista de los documentos entregados.

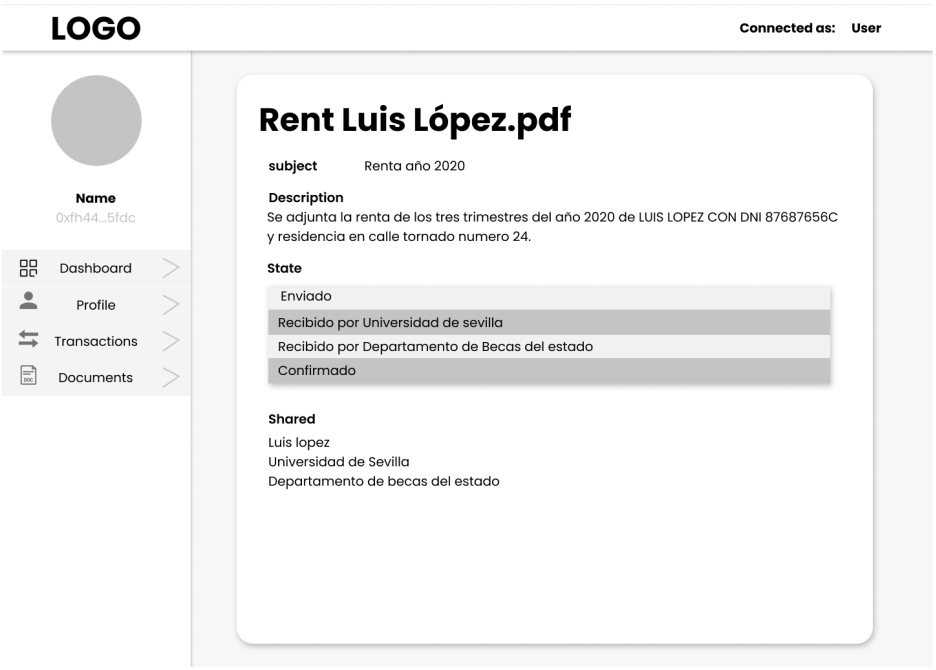

Figura 4.14: Vista detallada de un documento.

CAPÍTULO 5

Implementación y pruebas

En este capítulo, veremos cómo se ha llevado a cabo la implementación tras el análisis y el diseño del capítulo anterior. Para comenzar, se explicará el esquema tecnológico y cada una de sus herramientas en las Secciones 5.1, 5.2 y 5.3. En la Sección 5.4 se habla sobre la implementación y el código desarrollado, y por último, en la Sección 5.5 se explican las pruebas realizadas a la aplicación para su correcto funcionamiento.

5.1– Esquema tecnológico

En la siguiente figura 5.1, se muestra la relación de las tecnologías y herramientas que se han usado, pertenecientes al back-end,front-end y almacenamiento descentralizado.

Podemos destacar los proveedores. Metamask nos provee de una cartera y nos permite conectarnos con el back-end y la blockchain, e Infura, el cual nos permite conectarnos con el sistema de almacenamiento descentralizado IPFS.

Como se puede observar, dentro del back-end, está la blockchain de ethereum, la cual interactúa con la máquina virtual, en este caso creada por Ganache y Truffle, que conectan con el contrato inteligente escrito en Solidity.

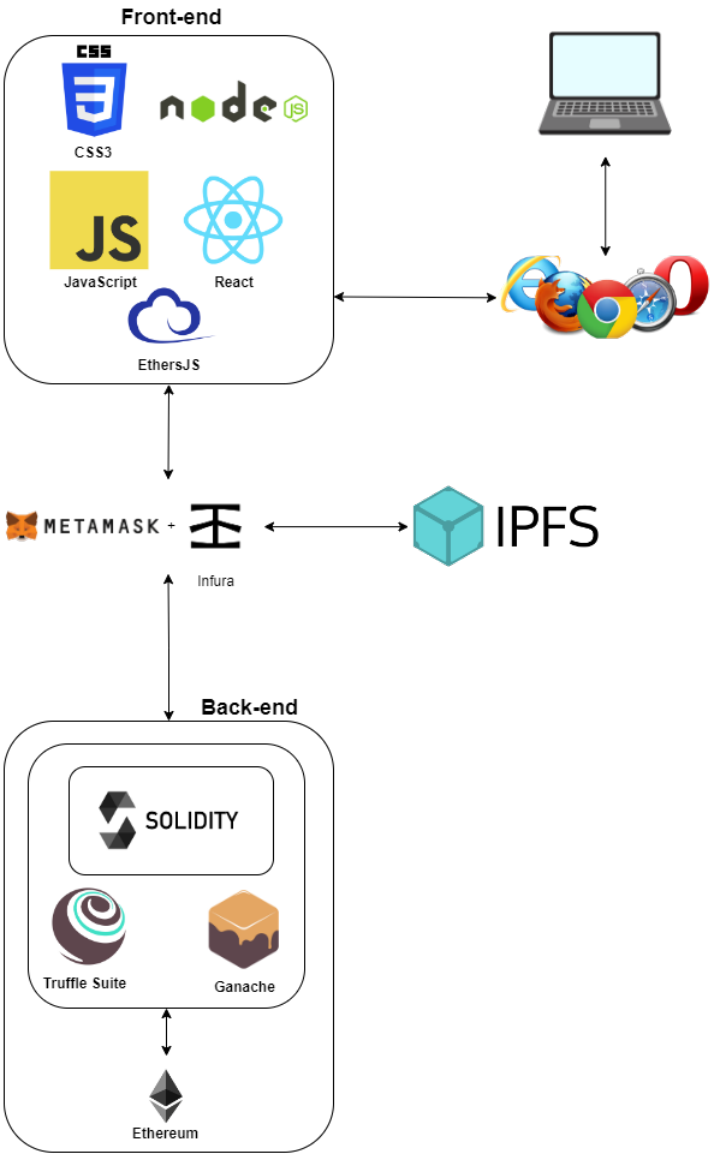

Figura 5.1: Diagrama de tecnologías usadas.

5.2– Arquitectura tecnológica

La arquitectura del sistema, ha sido descrita en el capítulo anterior, en la figura 4.4, la cual tendrá tantos servidores como nodos haya en la blockchain y en el sistema de almacenamiento distribuido. Las tecnologías usadas en front-end y back-end son las siguientes.

En el front-end se ha usado:

- **HTML5** lenguaje estándar para crear webs en la World Wide Web, basado en etiquetas.
- **CSS3** lenguaje de diseño gráfico usado para estilizar la web, usado junto a HTML5.

Figura 5.2: Logo de CSS3.

- **Javascript** como lenguaje de programación, ligero, orientado a objetos, usado comúnmente para crear aplicaciones webs y dotarlas de interactividad.
- **NodeJS** entorno en tiempo de ejecución basado en JavaScript, que funciona de manera asíncrona.
- **ReactJS** como librería del lenguaje JavaScript, usado para facilitar la creación de interfaces y de Single Page Applications o SPAs, haciendo uso de componentes y funciones.
- **Ethers.js** librería basada en JavaScript usada para interactuar con los contratos inteligentes desarrollados en la blockchain de Ethereum de manera simplificada.

Para el back-end se ha usado:

Figura 5.3: Logo de Javascript.

Figura 5.4: Logo de Node.JS.

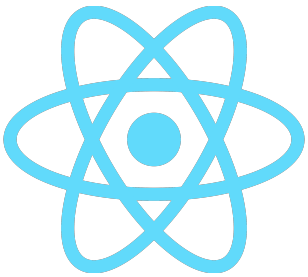

Figura 5.5: Logo de React.

Figura 5.6: Ethers JS.

- **Ganache**, una herramienta que permite desplegar una blockchain en local basada en Ethereum, diseñada especialmente para ejecutar

y probar contratos inteligentes, para acelerar y simplificar el proceso de desarrollo.

- **Truffle**, framework complementario a Ganache, que permite compilar y desplegar los contratos inteligentes de Ethereum entre otros, simplificando la ejecución de estos.

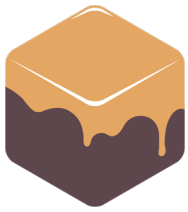

Figura 5.7: Ganache

Figura 5.8: Truffle

- **Solidity**, lenguaje de programación orientado a objetos para el desarrollo de contratos inteligentes compatibles con las blockchain que usan la máquina virtual de Ethereum. Basada en la sintaxis ECMAScript, pero centrado en la declaración de estructuras, las cuales son uno de los tipos más usados en los contratos inteligentes.

Figura 5.9: Solidity.

- **IPFS**, o Sistema de archivos interplanetario, es un protocolo y red peer-to-peer [24] creada para almacenar y compartir archivos en un sistema de almacenamiento descentralizado .

Figura 5.10: Solidity.

Para la conexión entre front-end y back-end, se ha usado:

- **Metamask**, extensión de navegadores y aplicación móvil, permite administrar claves de carteras criptográficas, transmitir y firmar transacciones, además de funcionar como billetera para enviar y recibir criptomonedas en varias blockchains.

Figura 5.11: Metamask.

5.3– Otras Herramientas

Para hacer más cómodo el desarrollo, se han usado diferentes herramientas.

- **Figma**, herramienta de edición de gráficos vectorial, basada en la web, usada especialmente en el prototipado previo de las vistas de una aplicación web. Se ha usado la versión gratuita de la aplicación web.

- **GitHub**, plataforma para alojar proyectos, que usa el sistema de control de versiones de Git. Se ha usado la versión PRO, gratuita al ser estudiante.

- **GitKraken**, cliente con interfaz de usuario gráfica, que permite gestionar los proyectos almacenados en GitHub y en local, de manera sencilla. Se ha usado la versión PRO, gratuita al ser estudiante.

- **Visual Studio Code**, editor de código fuente, con soporte para depuración, y la posibilidad de instalar extensiones que facilitan el desarrollo. Es gratuito, puesto que cuenta con licencia MIT.

- **Remix**, editor de código fuente en la web, para el desarrollo de smart contracts en Solidity. Tiene funcionalidades para poder desplegar y probar los contratos, además de compilar y depurar el código. Es gratuita.

- **Diagrams.net**, aplicación web que permite realizar todo tipo de diagramas. Es gratuita.
- **Infura**, conjunto de herramientas para facilitar el desarrollo en Web3 [25].

5.4– Detalles de la implementación

En esta sección, explicaremos la estructura de paquetes seguidas durante la implementación y puntos a destacar sobre el contenido de los elementos.

5.4.1. Estructura de paquetes

Los paquetes de la aplicación se han distribuido en dos carpetas principales, /Contracts y /Components.

Directorio Raíz

El directorio raíz, como se ve en la figura 5.12 es el primer nivel donde se encuentran las primeras carpetas. Dentro de /src hay 4 carpetas que se pueden ver en la figura 5.13, /Components y /Contracts que son las carpetas principales y serán explicadas más adelante, contamos también con una carpeta /Routes que gestiona el sistema de rutas de la aplicación, y otra /static con los componentes estáticos de la aplicación, como las imágenes.

la carpeta /public contiene los metadatos de la aplicación, como el nombre y el logo que se verán en el navegador.Los archivos package y package-lock , contienen en un JSON las dependencias que se usan en el proyecto que están contenidas en la carpeta /node_modules.

El resto de los archivos son archivos de configuración de Git, o módulos necesarios para su correcto funcionamiento.

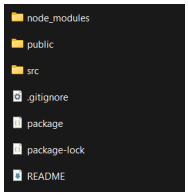

Figura 5.12: Distribución del directorio raíz.

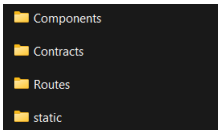

Figura 5.13: Distribución de src.

Paquete Contracts

Este paquete es el que contiene los elementos del back-end, es decir, todo lo relacionado con la blockchain. En la carpeta **contracts** están todos los contratos inteligentes desarrollados en Solidity, la carpeta **migrations** que contiene el código necesario para desplegar los contratos, la carpeta **test** que contiene las pruebas unitarias realizadas a los contratos inteligentes, y por último, el archivo truffle-config, el cual contiene la configuración de la herramienta mencionada anteriormente, Truffle, como la dirección local que se usa, que conectamos a Ganache para que todo funcione correctamente.

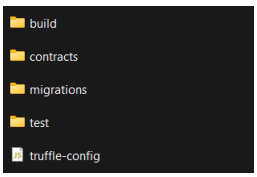

Figura 5.14: Paquete Contracts.

Paquete Components

Podemos observar en la figura 5.15 la estructura que sigue. Tenemos dos carpetas, y una gran cantidad de archivos JavaScript con extensión .js que hacen referencia a cada uno de los componentes usados para el desarrollo de la aplicación, en este caso de la vista de usuario. Todos estos archivos son elementos de React, intentando seguir una estructura de archivos lo más granulada posible, de manera que se puedan reutilizar funciones o componentes de unas vistas a otras, el cuál es uno de los principios de React.

Encontramos también una carpeta llamada **enterprise**, la cual contiene, al igual que en la carpeta Components, los componentes y vistas de la empresa.

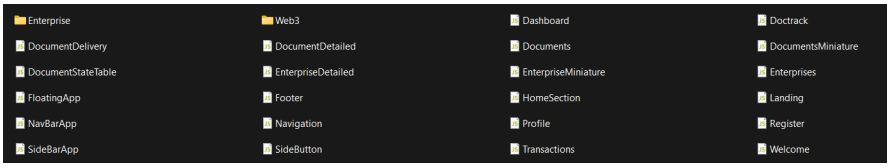

Figura 5.15: Paquete Components.

Por último, una carpeta llamada **Web3** en la figura 5.16, donde se alojan los archivos que realizan la conexión entre el front-end y los contratos inteligentes que desplegaremos en la blockchain. En estos archivos, de extensión .js se usa la librería de Ethers.js, explicada en la sección anterior, y se importan los archivos JSON de la carpeta build de Contracts, archivos que contienen toda la información sobre el despliegue en la blockchain, como puede ser la dirección de contrato que se le ha asignado, los archivos ABI o Application Binary Service [10] y el proveedor, los cuales se usarán para crear las funciones de llamada a cada función del contrato inteligente. Se puede observar un ejemplo en la figura 5.17.

Dentro de esta carpeta, también existe el archivo ipfs.js , el cual crea la instancia necesaria en infura para usar IPFS. Podemos ver la conexión en la figura 5.18.

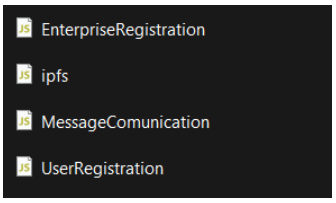

Figura 5.16: Carpeta Web3.

```
//contract details
const contractAdress = "0xB18b568a504341f9A75DAFfD92Fa2d53139D815d";
const abi = UserContract.abi;
const provider = new ethers.providers.Web3Provider(window.ethereum);
const UserContractDeployed = new ethers.Contract(contractAdress, abi, provider.getSigner());

//Contract functions
export async function addEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street){
    await UserContractDeployed.addEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street);
}
```

Figura 5.17: Conexión del contrato Enterprise.

```
const client = create({
    host: 'ipfs.infura.io',
    port: 5001,
    protocol: 'https',
    headers: {
        authorization: auth,
    },
});
```

Figura 5.18: Conexión de IPFS con Infura.

5.4.2. Código desarrollado

A continuación, se va a comentar algunos elementos destacables del código back-end y front-end, como los problemas encontrados y su solución.

Se procede a mostrar varios elementos destacables del código, incluyendo back-end y front-end, y algunos problemas que ocasionaron.

Se ha de destacar en el back-end, la llamada a funciones de parámetros de una estructura, como se ve en la figura 5.19. Aún con cursos y tutoriales realizados, los primeros pasos con Solidity fueron una toma de contacto real con el lenguaje. Se realizó en Visual Studio Code, y se probaban los contratos inteligentes en Remix, añadiendo parámetros especiales para la visibilidad entre funciones o actores que realizan estas funciones.

Un problema que se tuvo que remediar, fue el número máximo de llamadas a parámetros en una función, puesto que los bloques tienen una memoria máxima que no podemos superar. En nuestro caso, como nuestras estructuras son de un tamaño considerable, más de 10 variables a modificar, tuvimos que dividir algunas funciones en dos o más, puesto que existe un límite de memoria en las estructuras de solidity, que superabamos a partir de 12 variables, y crear un pequeño valor predeterminado al inicio.

```
function addEnterprise(
    address _wallet,
    string memory _name,
    string memory _businessName,
    string memory _fiscalNumber,
    string memory _phone,
    string memory _email,
    string memory _country,
    string memory _region,
    string memory _postalCode,
    string memory _city,
    string memory _street
) public {
    Enterprise memory enterprise = Enterprise(
        _wallet,
        _name,
        _businessName,
        _fiscalNumber,
        0,
        _phone,
        _email,
        _country,
        _region,
        _postalCode,
        _city,
        _street,
        "QmUso3hDnd4X14yM358fV7H4oqTqhh9sz18s2sH1QhLVW5"
    );
    enterprises[_wallet] = enterprise;
    enterprisesWalletArray.push(_wallet);
}
```

Figura 5.19: Función añadir empresa.

Solidity tiene un tipo de función llamada modificador, la cual como su nombre indica, la modifica para, normalmente restringirla y más tarde se añaden a otras funciones en su creación. De esta manera, si se busca que solo una empresa llame a la función, o que la empresa exista para realizar la función, solo hay que añadir una palabra. Se muestra un ejemplo en la

figura 5.20.

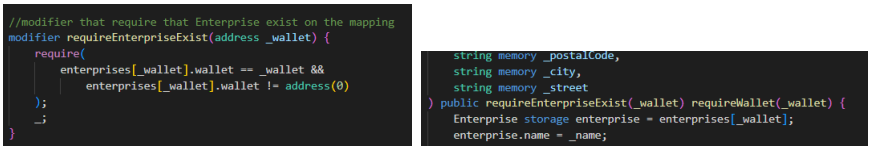

Figura 5.20: Función modificadora y llamada en función.

Vamos a explicar ahora un contrato inteligente al completo. Para ello usaremos como referencia el contrato inteligente de empresa, apoyado en la figura 5.21, figura 5.22 y figura 5.23.

En la figura 5.21 se declaran todas los parámetros que tendrá la empresa, con su tipo, dentro de una estructura, la cual representa el tipo empresa. Justo al finalizar la estructura, creamos un mapeo por el que cada dirección se convertirá en una estructura empresa, y el mapeo completo será la lista de empresas. Por último habrá una lista de direcciones que contendrá a todas las empresas que se creen dentro del contrato.

Tras ello, la primera función se puede ver en la figura 5.19 es la de añadir una empresa a la blockchain. Para ello, recibe los parámetros y se crea una estructura empresa con los datos recibidos, para más tarde añadirlos al mapeo y a la lista.

Las funciones que se pueden ver en la figura 5.22 son para actualizar los datos, los cuales obtienen la empresa a través de la dirección unica y actualizan uno o varios parámetros de la estructura. En la función "updateImageHash" intervienen dos modificadores, que se pueden ver definidos en la figura 5.23. La función "getEnterprise" a partir de una wallet, devuelve la estructura completa, esta función contiene la palabra view, la cual solo hace petición a la blockchain y no escribe ningún tipo de dato, por lo que no será necesario firmar ninguna transacción para llamar a la función, al contrario si no contiene este modificador.

Por último, podemos observar como se escriben los modificadores en la figura 5.23. En este caso, por ejemplo el modificador "requireEnterpriseExist" pide como condición que al buscar dentro del mapa de empresas debe devolver una dirección no nula. y la _ del final representa el código que se realizará si se cumple el modificador en la función.

Vamos a proceder a mostrar cómo se llaman a las funciones de los smart contracts desde el front-end.

Como se observa en la figura 5.24, se importa el JSON del contrato, el cual se genera tras desplegarlo en la blockchain. Declaramos el abi que contiene el JSON y la dirección de contrato que podemos obtener en el despliegue del contrato. También obtenemos el proveedor, y con estos tres datos, se crea el contrato para el front-end.

El siguiente paso es declarar las mismas funciones que tenemos en el

```
contract EnterpriseContract is Ownable {
    struct Enterprise {
        address wallet;
        string name;
        string businessName;
        string fiscalNumber;
        int8 minimumLevel;
        string phone;
        string email;
        string country;
        string region;
        string postalCode;
        string city;
        string street;
        string imageHash;

    }

    mapping(address => Enterprise) private enterprises;
    address[] public enterprisesWalletArray;
```

Figura 5.21: Declaración variables de empresa.

```
    function updateMinimumLevel(address _wallet, int8 _minimumLevel) public {
        Enterprise storage enterprise = enterprises[_wallet];
        enterprise.minimumLevel = _minimumLevel;
    }

    function updateImageHash(address _wallet, string memory _imageHash)
        public
        requireEnterpriseExist(_wallet)
        requireWallet(_wallet)
    {
        Enterprise storage enterprise = enterprises[_wallet];
        enterprise.imageHash = _imageHash;
    }

    //TODO: GET Enterprise only if same wallet or enterprise
    function getEnterprise(address _wallet)
        public
        view
        returns (Enterprise memory enterprise)
    {
        enterprise = enterprises[_wallet];
    }
```

Figura 5.22: Funciones actualizar datos de empresa.

```
    /* function to get all enterprises from mapping */
    function getAllEnterprises()
        public
        view
        returns (Enterprise[] memory enterpriseList)
    {
        enterpriseList = new Enterprise[](enterprisesWalletArray.length);
        for (uint256 i = 0; i < enterprisesWalletArray.length; i++) {
            address wallet = enterprisesWalletArray[i];
            enterpriseList[i] = enterprises[wallet];
        }
        return enterpriseList;
    }

    //modifier that require that Enterprise exist on the mapping
    modifier requireEnterpriseExist(address _wallet) {
        require(
            enterprises[_wallet].wallet == _wallet &&
                enterprises[_wallet].wallet != address(0)
        );
        _;
    }

    //modifier to check that wallet is the same as the one in the mapping
    modifier requireWallet(address _wallet) {
        require(msg.sender == _wallet);
        _;
    }
}
```

Figura 5.23: Funciones modificadoras de empresa.

contrato inteligente de Solidity, de manera asíncrona y con los parámetros necesarios.

Para llamarlo desde el front-end, debemos importar este archivo al componente y hacer la llamada a la función necesaria, y, de esta manera, escribimos y recuperamos datos en la blockchain.

```
import UserContract from '../../Contracts/build/contracts/EnterpriseContract.json';
import { ethers } from 'ethers';

//contract details
const contractAdress = "0xB18b568a504341f9A75DAFfD92Fa2d53139D815d";
const abi = UserContract.abi;
const provider = new ethers.providers.Web3Provider(window.ethereum);
const UserContractDeployed = new ethers.Contract(contractAdress, abi, provider.getSigner());

//Contract functions
export async function addEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street){
    await UserContractDeployed.addEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street);
}

export async function getEnterprise(wallet){
    const enterprise = await UserContractDeployed.getEnterprise(wallet);
    return enterprise;
}

export async function updateEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street){
    await UserContractDeployed.updateEnterprise(wallet, name, businessName, fiscalNumber, phone, email, country, region , postalcode, city , street);
}

export async function updateMinimumLevel(wallet, level){
    await UserContractDeployed.updateMinimumLevel(wallet, level);
}
```

Figura 5.24: Funciones modificadoras de empresa.

Otro punto importante es la conexión al sistema de almacenamiento descentralizado IPFS. Para ello, el primer paso es conectarnos a un nodo. En este caso, usamos el nodo de Infura [17] el cual se conecta tras crear un proyecto y declarar el ID y la clave secreta del proyecto en nuestra aplicación como se ve en la figura 5.25. Tras subir nuestro archivo a la aplicación, IPFS nos generará un hash que subiremos a la blockchain, actualizando los datos como hemos visto previamente, y para pedir la imagen tenemos que usar el subdominio de nuestro proyecto, junto con el hash creado, como se puede ver en la figura 5.26.

```
const auth =
    'Basic ' + Buffer.from(projectId + ':' + projectSecret).toString('base64');

//const ipfsClient = require('ipfs-http-client');
const client = create({
    host: 'ipfs.infura.io',
    port: 5001,
    protocol: 'https',
    headers: {
        authorization: auth,
    },
});
```

Figura 5.25: Conexión con el nodo para usar IPFS.

El front-end, como se ha mencionado en secciones anteriores, se ha desarrollado con React y JavaScript, para así poder modularizar y simplificar el desarrollo de las vistas, reutilizando alguno de sus componentes.

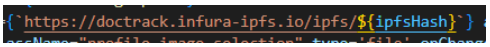

Figura 5.26: Obtener archivo de IPFS.

Un ejemplo puede ser el componente de Empresa o el de documento, los cuales simplemente llaman a un componente y se rellenan con los datos obtenidos en la blockchain.

Otra de las propiedades que nos provee React son los "Hooks"de efecto y estado, como son las funciones useEffect() y useState(), las cuales han permitido actualizar los componentes en tiempo real, y transmitir alguna información actualizada a las nuevas vistas.

Podemos ver un ejemplo en las figuras 5.27 y 5.28. La función useState nos permite escribir y leer las variables declaradas en tiempo real en nuestra aplicación. Y la función useEffect, se realiza siempre que un valor determinado cambia, y al renderizar la vista. Por ello, en la figura 5.28 se llama a la función loadProfile la cual a través de la cartera del usuario, pide los datos a la blockchain y los carga en las variables, que después se muestran por pantalla al usuario. Se puede observar un ejemplo en la figura 5.29.

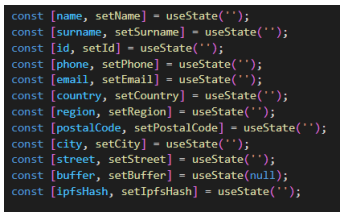

Figura 5.27: Ejemplo de uso de UseState.

React es un framework de SPA o single Page Application, por lo que no debería existir más que una página, pero otra de las propiedades usadas de React es la llamada router, la cual simula las diferentes rutas que usará la aplicación, para así poder redirigir al usuario al punto donde se desea en todo momento. De esta manera existen /landing , /docktrack y /enterprise las cuales dividen a la aplicación en las 3 vistas qe después se renderizan y cambian según la sección en la que el usuario se encuentre.

```
function loadProfile(wallet) {

    getUser(wallet).then(function (user) {
        setName(user.name);
        setSurname(user.surname);
        setId(user.id);
        setPhone(user.phone);
        setEmail(user.email);
        setCountry(user.country);
        setRegion(user.region);
        setPostalCode(user.postalCode);
        setCity(user.city);
        setStreet(user.street);
        setIpfsHash(user.imageHash);
    });

}


useEffect(() => {
    loadProfile(wallet);

}, []);
```

Figura 5.28: Ejemplo de uso de UseEffect.

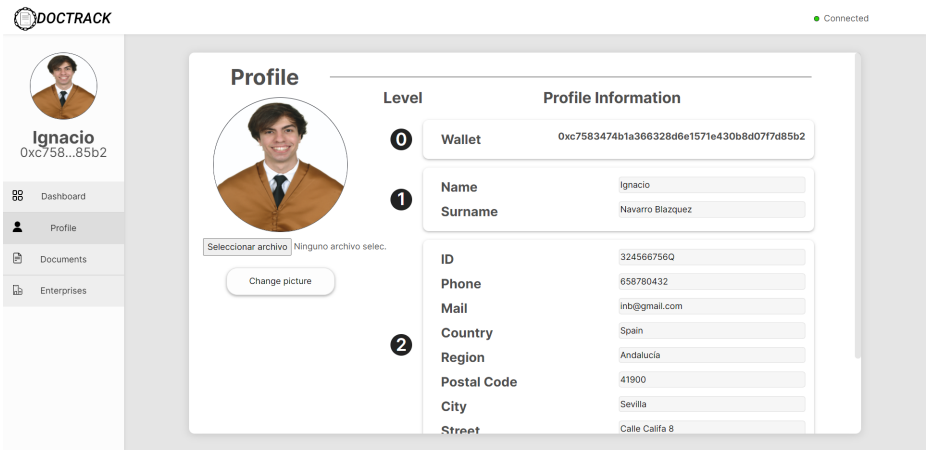

Figura 5.29: Mostrar datos de la blockchain por pantalla.

5.5– Pruebas

5.5.1. Pruebas unitarias de contrato inteligente

Los contratos inteligentes que se usan en el proyecto son una de las partes más críticas del sistema, por lo que es importante comprobar sus funcionalidades y restricciones, para que nada falle. En el IDE Remix, mencionado previamente en la Sección 5.3, hemos estado comprobando que, tras llamar a la función, los valores y los actores que pueden verlo son correctos, con acciones que no están permitidas, como por ejemplo, que un usuario que no ha mandado ni recibido el mensaje pueda verlo.
Pero estas pruebas están realizadas por humanos y pueden no ser correctas completamente, por ello, se realizaron también pruebas unitarias de las funciones más críticas de los diferentes contratos inteligentes, comprobando así que funcionan correctamente.

Estas pruebas han sido desarrolladas en JavaScript, y se realizan justo antes del despliegue, de manera que, si hay alguna errónea, se cancela el despliegue puesto que algo está fallando.

En la figura 5.30 se muestra un ejemplo de código para una prueba unitaria, y en la figura 5.31 se muestra el conjunto de pruebas, con el tiempo de ejecución y si han sido satisfactorias.

```
it('should set a message as confirmed', async () => {
    await instance.writeMessage(accounts[0], accounts[1], "Mensaje",
    "Cueerpo", "0x7234y734hkjh");
    await instance.setConfirmed(0);
    const message2 = await instance.getMessage(0);
    assert(message2.confirmed === true, "Message not different");
});
```

Figura 5.30: Ejemplo de prueba unitaria.

5.5.2. Pruebas de aceptación

Para todos los casos de uso de la Sección 4.2 se han realizado prueba de aceptación. Se realizará una explicación de cada uno de los test realizados y el resultado. También incluirá excepciones si existe alguna.

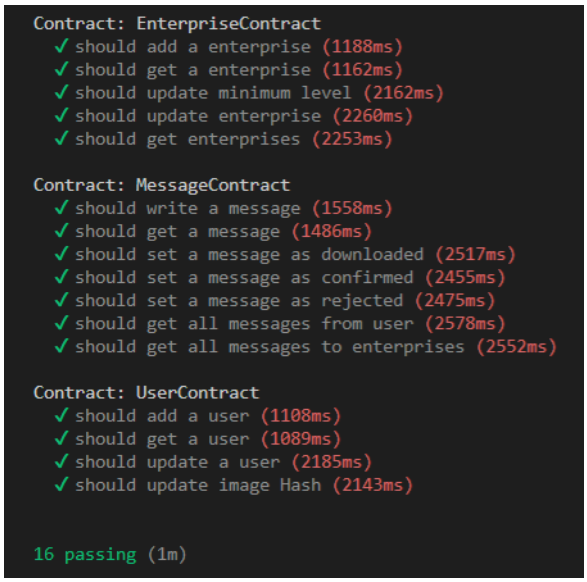

Figura 5.31: Resultados pruebas unitarias.

Caso de uso 1: Ver página de login de usuario

Test < #001 >	
Descripción	Comprobar que se muestra la pantalla de login de usuario a un usuario no autenticado.
Resultado	Se muestra la pantalla de login para usuarios al entrar en la aplicación, que permite iniciar sesión con la billetera o registrarse.

Tabla 5.1: Prueba de ver pantalla de login de usuario.

Caso de uso 2: Ver página de login de empresa

Test < #002 >	
Descripción	Comprobar que se muestra la pantalla de login de empresa a un usuario no autenticado.
Resultado	Se muestra la pantalla de login para empresas al entrar en la aplicación, que permite iniciar sesión con la billetera o registrarse.

Tabla 5.2: Prueba de ver pantalla de login de empresa.

Caso de uso 3: Registro de usuario

Test < #003 >	
Descripción	Comprobar que se crea una cuenta de usuario.
Resultado	Cuando un usuario no autenticado se registra en el sistema, la billetera le pedirá firmar la transacción y guardar los datos introducidos en el formulario en la blockchain.
Excepciones	No se rellenan los campos obligatorios o se rellenan incorrectamente.

Tabla 5.3: Prueba de registro un usuario.

Caso de uso 4: Registro de empresa

Test < #004 >	
Descripción	Comprobar que se crea una cuenta de empresa.
Resultado	Cuando un usuario no autenticado se registra como empresa en el sistema, la billetera le pedirá firmar la transacción y guardar los datos introducidos en el formulario en la blockchain.
Excepciones	No se rellenan los campos obligatorios o se rellenan incorrectamente.

Tabla 5.4: Prueba de registro de empresa.

Caso de uso 5: Ver perfil de usuario

Test < #005 >	
Descripción	Comprobar que se muestra el perfil del usuario y su información correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de perfil, la aplicación renderiza sus datos, obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.5: Prueba de vista de perfil de usuario.

Caso de uso 6: Ver perfil de empresa

Test < #006 >	
Descripción	Comprobar que se muestra el perfil de la empresa y su información correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de perfil de empresa, la aplicación renderiza los datos, obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.6: Prueba de vista de perfil de empresa.

Caso de uso 7: Editar foto de perfil

Test < #007 >	
Descripción	Comprobar que un usuario o empresa puede editar su foto de perfil.
Resultado	Cuando un usuario edite su imagen de perfil, la billetera pedirá firmar la transacción y confirmar la nueva imagen de perfil.
Excepciones	La herramienta muestra un aviso de error cuando: • No exista nueva foto de perfil.

Tabla 5.7: Prueba de edición de imagen de perfil.

Caso de uso 8: Editar información de perfil

Test < #008 >	
Descripción	Comprobar que un usuario o empresa puede editar su información de perfil.
Resultado	Cuando un usuario edite su información, la billetera pedirá firmar la transacción y confirmar los nuevos datos.
Excepciones	No se rellenan los campos obligatorios o se rellenan incorrectamente.

Tabla 5.8: Prueba de edición de información de perfil.

Caso de uso 9: Ver dashboard de usuario

Test < #009 >	
Descripción	Comprobar que se muestra la dashboard del usuario correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de dashboard, la aplicación renderiza la nueva vista,con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.9: Prueba de vista de dashboard de usuario.

Caso de uso 10: Ver dashboard de empresa

Test < #010 >	
Descripción	Comprobar que se muestra la dashboard de la empresa correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de dashboard de empresa, la aplicación renderiza la nueva vista,con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.10: Prueba de vista de dashboard de empresa.

Caso de uso 11: Ver documentos enviados

Test < #011 >	
Descripción	Comprobar que se muestra los documentos enviados del usuario correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de documentos, la aplicación renderiza la nueva vista, con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.11: Prueba de vista de documentos del usuario.

Caso de uso 12: Ver documentos recibidos

Test < #012 >	
Descripción	Comprobar que se muestra los documentos recibidos de la empresa correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de documentos de empresa, la aplicación renderiza la nueva vista, con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.12: Prueba de vista de documentos de la empresa.

Caso de uso 13: Ver la información de un documento

Test < #013 >	
Descripción	Comprobar que se muestra la información de un documento correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre uno de los documentos, la aplicación renderiza la nueva vista, con datos obtenidos directamente de la blockchain.
Excepciones	Si no se han enviado o recibido documentos, no habrá ninguno para pulsar.

Tabla 5.13: Prueba de vista de información de documento.

Caso de uso 14: Descargar un documento

Test < #014 >	
Descripción	Comprobar que se puede descargar el documento seleccionado por el usuario, del mensaje en el que se encuentra.
Resultado	Cuando un usuario desde la vista de empresa pulse sobre descargar documento, la billetera pedirá firmar la transacción y con ello mostrará el documento, a la vez que actualiza el estado en la blockchain.
Excepciones	En el caso de que sea en la vista de usuario donde se intenta esta función, no habrá que firmar ninguna transacción ya que no se actualiza el estado.

Tabla 5.14: Prueba de descarga de documento.

Caso de uso 15: Confirmar un documento

Test < #015 >	
Descripción	Comprobar que se puede confirmar el documento seleccionado por el usuario, del mensaje en el que se encuentra.
Resultado	Cuando un usuario desde la vista de empresa pulse sobre confirmar documento, la billetera pedirá firmar la transacción y actualizar el estado en la blockchain.
Excepciones	La aplicación no confirmará el documento cuando: • Ya esté confirmado. • Esté rechazado.

Tabla 5.15: Prueba de confirmación de documento.

Caso de uso 16: Rechazar un documento

Test < #016 >	
Descripción	Comprobar que se puede rechazar el documento seleccionado por el usuario, del mensaje en el que se encuentra.
Resultado	Cuando un usuario desde la vista de empresa pulse sobre rechazar documento, la billetera pedirá firmar la transacción y actualizar el estado en la blockchain.
Excepciones	La aplicación no rechazará el documento cuando: • Ya esté rechazado. • Esté confirmado.

Tabla 5.16: Prueba de retirar usuario del activo.

Caso de uso 17: Ver empresas registradas

Test < #017 >	
Descripción	Comprobar que se muestra la lista de empresas disponibles correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre el botón de empresas, la aplicación renderiza la nueva vista, con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.17: Prueba de ver dashboard de la organización.

Caso de uso 18: Ver información sobre la empresa

Test < #018 >	
Descripción	Comprobar que se muestra la información de la empresa seleccionada correctamente en la aplicación.
Resultado	Cuando un usuario pulsa sobre la empresa, la aplicación renderiza la nueva vista, con datos obtenidos directamente de la blockchain.
Excepciones	Si no está logueado no podrá acceder.

Tabla 5.18: Prueba de vista de información sobre una empresa.

Caso de uso 19: Enviar documento a empresa

Test < #019 >	
Descripción	Comprobar que se envía un documento de un usuario a una empresa correctamente.
Resultado	Cuando el usuario envía un documento, la billetera pedirá firmar la transacción y con ello enviará el documento, a la vez que se suben los datos a la blockchain.
Excepciones	La herramienta muestra un aviso de error cuando: • No se rellenan los campos obligatorios o se rellenan incorrectamente. • Un administrador intenta crear el comentario.

Tabla 5.19: Prueba cerrar sesión.

CAPÍTULO 6

Conclusiones

Este capítulo contendrá las conclusiones obtenidas tras todo el desarrollo del proyecto. Comenzaremos por un análisis retrospectivo, con los aspectos que se han hecho bien y los que se podrían mejorar en próximos proyectos en la Sección 6.1. Continuaremos con los conocimientos obtenidos al desarrollar el proyecto, en la Sección 6.2. Por último, se expondrán posibles mejoras de cara al futuro para la aplicación desarrollada en la Sección 6.3.

6.1– Retrospectiva

6.1.1. Aspectos a repetir

La planificación del proyecto ha sido muy acertada, con menos de un 5 % de diferencia entre el tiempo de desarrollo estimado y el real. Por lo cual no ha habido problemas de tiempo en el desarrollo. Gracias a ello, se ha podido crear un sistema de envío de documentos funcional y con una estructura que permite incluir mejoras al sistema sin romperlo.

La comunicación entre tutor y alumno ha sido fluida, lo cual ha ayudado a que el proyecto no estuviera estancado en ningún momento.

El previo desarrollo y diseño de maquetas o mockups y el tener una idea clara, ha ayudado a que la aplicación tenga un estilo homogéneo y bonito que comparten todas las vistas.

6.1.2. Aspectos a mejorar o evitar

El mayor problema ha sido el uso de nuevas tecnologías, puesto que una gran parte del tiempo se ha consumido en aprender, y corregir errores que estas ocasionaban. El mayor retraso se ha generado por paquetes y versiones que no eran compatibles entre ellas, pero tras el análisis y la corrección de las versiones, los problemas encontrados eran solucionables mediante una búsqueda en internet, puesto que React es un lenguaje muy usado y Solidity cuenta con una gran comunidad de desarrolladores detrás.

Sin duda una mayor preparación y práctica, hubiera supuesto menos problemas a la hora del desarrollo.

6.2– Lecciones aprendidas

La planificación es una de las principales causas en los proyectos fallidos o mal elaborados, por lo que consideramos y hemos visto de primera mano que una planificación en condiciones puede ser el diferencial en el momento de desarrollar un proyecto ya sea software o de otro ámbito.

Es muy necesario también, realizar un buen análisis del sistema a desarrollar y realizar diseños previos, ya que facilita su posterior desarrollo. Siguiendo con posibles acciones recomendadas, la comunicación con las posibles partes del proyecto es clave, ya que sirve para mitigar los posibles fallos o incoherencia con el cliente final, reduciendo la posible tasa de proyecto fallido.

Solidity es un lenguaje que se usa en muchas blockchains puesto que se ejecuta con la EVM, y las blockchain han añadido compatibilidad con esta en la mayoría de los casos, puesto que es donde más contratos inteligentes hay escritos y que más uso se les dan. Por ello, el uso en entorno real es muy probable si se decidiera desplegar en cualquiera de las blockchains compatibles. Pero hay que tener en cuenta el gasto por despliegue y firma de los contratos, por lo que lo más recomendable sería desplegarlo en una capa 2 o de bajo coste como podría ser Arbitrum o Polygon redes que tienen tasas incluso menores de 1 céntimo, frente a las tasas de Ethereum, que, dependiendo de la congestión de la red pueden oscilar entre 4 y 30 dólares.

Destacar finalmente que, gracias al desarrollo del proyecto, hemos acabado aprendiendo tanto Solidity como React. Tecnologías muy demandadas en la actualidad, y que, gracias a los conocimientos adquiridos, podrían suponer oportunidades laborales, o incluso nuevos proyectos en solitario.

6.3– Posibles mejoras del sistema

Este proyecto, aunque es funcional completamente, tiene un alcance muy grande, por lo que se le pueden añadir muchos tipos de mejoras y nuevas funcionalidades.

Un gran problema a lo largo del tiempo para esta aplicación es el coste que tiene desplegarse en la red de Ethereum, puesto que es una de las más costosas. Sin embargo, existen alternativas o segundas capas [19] que son compatibles con la EVM y que por tanto no necesitarían modificación de código para funcionar en estas blockchains, o incluso una blockchain privada, donde se asumieran los costes de transacción para los usuarios.

Otra de las funcionalidades que se pueden añadir es el KYC o Know Your Customer [23], para añadir un nivel de seguridad extra y confirmar que los usuarios finales son quienes dicen ser, ya que para este proceso se necesita subir un archivo con tu DNI y ser confirmado externamente.

Añadir un nuevo actor, que represente los departamentos de la empresa y que solo la empresa pueda crear, y que la empresa pueda enviar los documentos de sus usuarios a los departamentos pertinentes, obteniendo así una mejor organización y eficacia.

Añadir marcas de tiempo al estado del documento, puesto que la actualización de los estados ahora mismo no la tiene, y puede agregar mucho valor a la solución para saber los momentos exactos en los que la empresa empieza a revisar el documento enviado.

Y como este, también hay pequeñas funcionalidades que agregan bastante valor como:

- Ordenar los documentos recibidos/enviados.
- Pop ups con mensajes al realizar alguna acción.
- hacer el dashboard inteligente según usuario.

CAPÍTULO 7

Manuales

7.1– Manual de instalación y despliegue

El manual explica los pasos y programas necesarios para ejecutar la aplicación para Windows 10/11.

7.1.1. Instalación requisitos back-end

El primer paso para poder desplegar nuestra aplicación, debemos instalar NodeJS. Para ello, nos dirigiremos a `https://nodejs.org/es/download/releases/` y buscamos la versión 16.15.0 Gallium, que contiene la versión 8.5.5 de NPM. Gracias a la cual podremos instalar los paquetes que se han usado en el desarrollo del proyecto y que sin ellos no funcionarían.

Al descargar, abriremos el instalador y seleccionamos todos los ajustes por defecto.

Procedemos ahora a instalar Ganache [15], herramienta que nos proporciona una blockchain de prueba en local, y donde podremos desplegar y visualizar los diferentes contratos inteligentes que usa nuestra aplicación. La usada durante el proyecto ha sido la versión 2.5.4 que podemos encontrar en el siguiente enlace `https://github.com/trufflesuite/ganache-ui/releases/tag/v2.5.4`. Tras la descarga abriremos el instalador y dejamos los ajustes por defecto.

Por último, instalaremos también un paquete llamado Truffle, que complementa a Ganache y con la cual podremos hacer test y desplegar los contratos en la red de prueba de Ganache. Para ello, entraremos en el directorio de nuestra aplicación y abrimos una consola de comandos para escribir lo siguiente.

```
1       cd src
2       cd Contracts
3       npm install -g truffle
```

Código 7.1: Instalación de Truffle

Tras la instalación de las herramientas, vamos a preparar el entorno para que pueda ejecutar nuestra aplicación sin problemas.

Abriremos Ganache y pulsamos sobre "New Workspace Ethereum" el cuál nos dotará de un espacio de pruebas siempre igual, al contrario que con "QuickStart Ethereum". Podemos dejar los valores que se encuentran por defecto puesto que más tarde entraremos a la configuración. Aunque, se recomienda añadir una cuenta con su frase semilla, nos ahorrará pasos más adelante.

Una vez iniciado, pulsaremos en el icono de los ajustes que se encuentra arriba a la derecha. Para empezar, deberíamos añadir nuestro truffle-config.js a nuestro entorno como se ve en la figura 7.1.

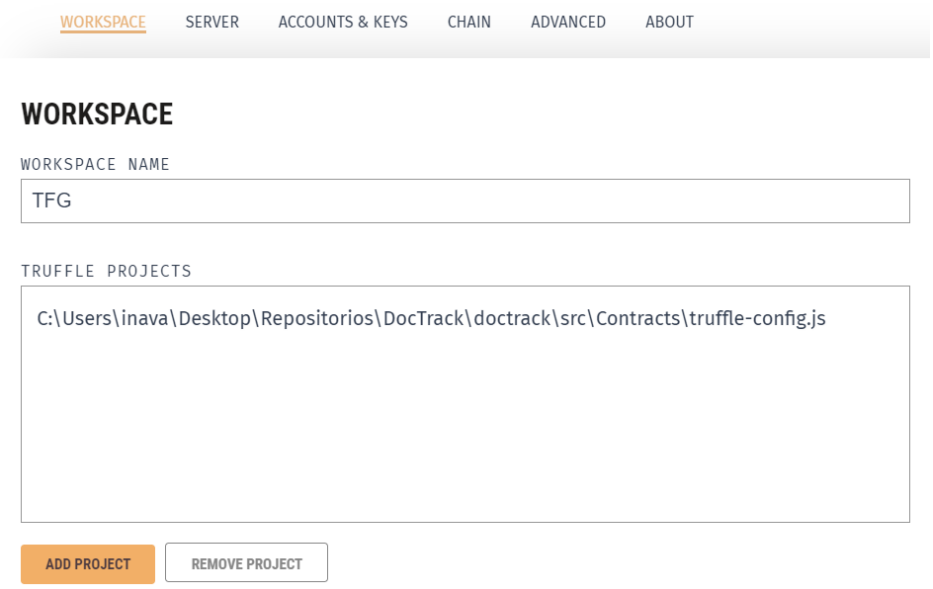

Figura 7.1: Información del workspace.

Tras ello, pasamos a la pestaña de "Server" y deberíamos tenerlo de la siguiente manera, como se ve en la figura 7.2:

Comprobamos, abriendo nuestro truffle-config.js dentro de la carpeta /Contracts que los valores de la línea 44, 45 y 46, Host, Port y Network Id,

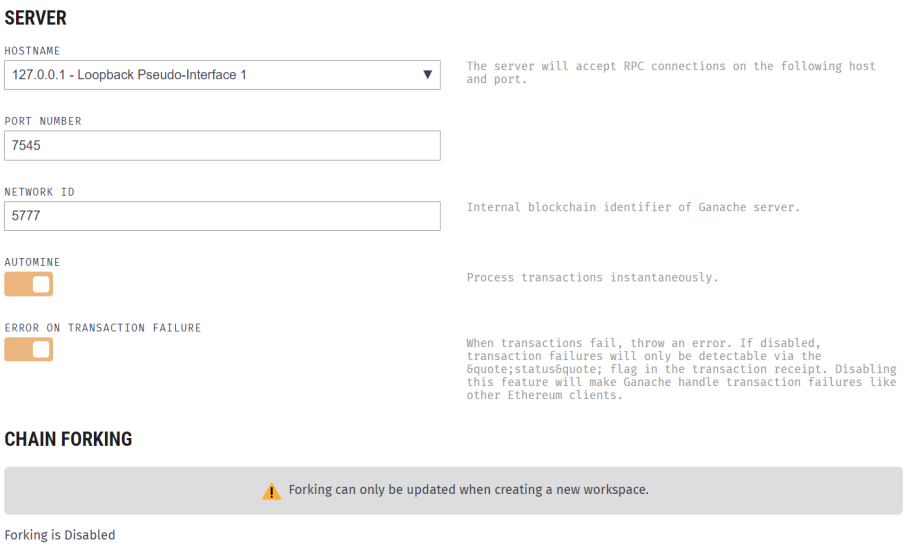

Figura 7.2: Información del servidor.

son como se ve en la siguiente imagen, pero al ser los valores por defecto no necesitan estar sin comentarios.

```
networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache, geth, or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
  // development: {
  //  host: "127.0.0.1",     // Localhost (default: none)
  //  port: 8545,            // Standard Ethereum port (default: none)
  //  network_id: "*",       // Any network (default: none)
  // },
```

Figura 7.3: Valores por defecto truffle-config.js.

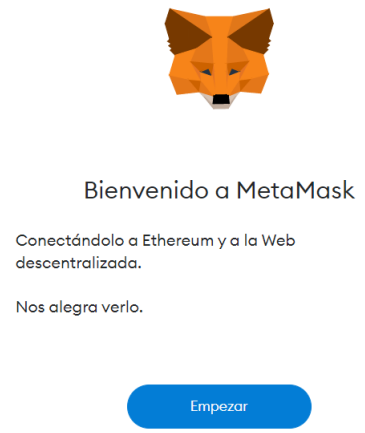

Figura 7.4: Página de bienvenida en Metamask.

7.1.2. Instalación requisitos front-end

Como se explicó al instalar NPM, son necesarios módulos y paquetes para que nuestra aplicación funcione correctamente. Para ello, iremos al directorio raíz de nuestro código, el cuál es el que está contenido en doctrack, que contiene las carpetas /src, /public y el resto de archivos, y ejecutaremos el siguiente comando en la consola:

```
1 npm install
```

Código 7.2: Instalación dependencias de Node

Necesitaremos la herramienta que conecta nuestro front-end con nuestro back-end, en este caso la blockchain. Para ello, usaremos Metamask [16]. Para su instalación entraremos en la Chrome Web Store y buscamos Metamask, o podemos hacerlo desde el siguiente enlace: `https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn`.

Tras añadirlo, se nos abrirá la pantalla de bienvenida y pulsamos en empezar como se ve en la siguiente figura 7.4. Si Por algún casual, no se abriera, simplemente debemos ir a nuestras extensiones y pulsar sobre esta.

Si se están realizando estos pasos antes de preparar Ganache, deberemos pulsar en “Crear cartera“ y seguir los pasos necesarios hasta su finalización, y más tarde en la instalación de Ganache, añadir la frase

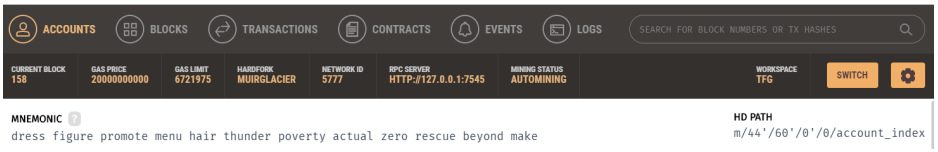

Figura 7.5: Frase semilla de Ganache.

semilla mientras creamos nuestro entorno.

Si, por el contrario, hemos instalado Ganache primero, pulsaremos sobre Importar cartera, ya que Ganache nos proporciona una frase semilla al crear el entorno. Para obtenerla, simplemente abriremos Ganache y seleccionaremos nuestro entorno ya creado, y en la pestaña de Accounts copiaremos las 12 palabras que se encuentras en "MNEMONIC" como se ve en la figura 7.5.

Estas 12 palabras en inglés son la frase semilla, que dan acceso a nuestra cartera a cualquier persona que cuente con ellas, y por tanto hacer transacciones y firmar desde ella.

Pegamos las 12 palabras en el orden en el que se nos pide, o todas en la primera casilla y se nos rellenará el resto con las demás palabras de la frase. Por último, nos pedirá una contraseña para acceder y ya tendríamos la billetera importada.

Ahora, seleccionaremos la red de pruebas que usamos en Ganache, ya que Metamask, por defecto, usa la red principal de Ethereum. Para ello, si hemos cerrado la pestaña, pulsaremos sobre nuestra extensión y tendríamos la vista simplificada de la pestaña. Buscamos el botón en el que pone "Show/Hide test networks".

Por último, iremos a la pestaña de "Redes" y modificaremos el puerto de la URL de RPC de LocalHost para que quede como en la figura 7.8. Aquí puede darnos algún error, pero simplemente reintentándolo se puede conseguir.

Seleccionaremos la red Localhost y ya estaremos conectados a Ganache.

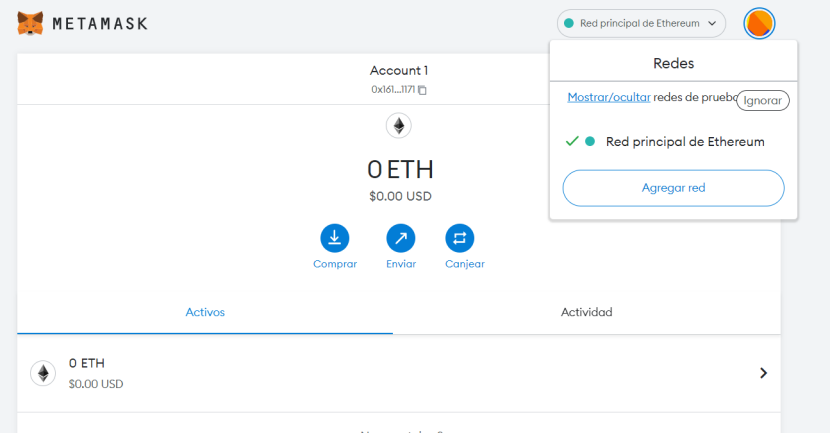

Figura 7.6: Añadir redes de prueba a Metamask.

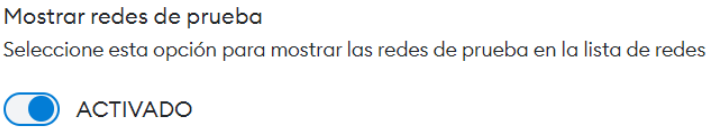

Figura 7.7: Activar redes de prueba.

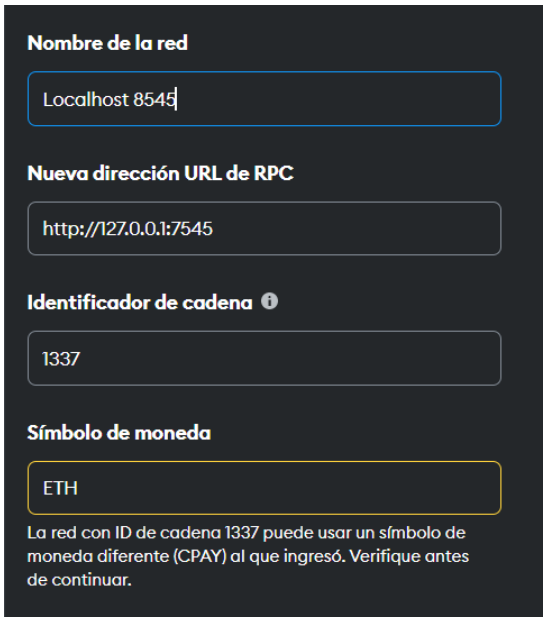

Figura 7.8: Red Localhost actualizada.

7.1.3. Despliegue contratos inteligentes

Previo a poder desplegar la aplicación, debemos desplegar los contratos inteligentes desarrollados en nuestra red de pruebas, para poder interactuar con ellos, almacenando y obteniendo los datos almacenados. Para ello, abriremos una consola de comandos desde el directorio raíz y escribimos lo siguiente:

```
1    cd src
2    cd Contracts
3    truffle migrate
```

Código 7.3: Despliegue contratos inteligentes

Podremos ver que se han desplegado correctamente, puesto que la consola nos indica el coste de despliegue, así como la dirección para interactuar con ellos. También podremos ver en Ganache, en la pestaña "Contracts" los que están desplegados puesto que tienen la etiqueta "Deployed".

7.1.4. Despliegue sistema final

Para desplegar correctamente nuestra aplicación, simplemente escribimos desde el directorio raíz en una consola de comandos lo siguiente:

```
1    npm run start
```

Código 7.4: Ejecutar servicio front-end

Podremos observar, como tras una breve espera, se nos abre una nueva pestaña en el navegador con la dirección `http://localhost:3000/` y una ventana emergente de Metamask pidiendo la contraseña, y conectar la billetera a la aplicación, mensaje que debemos aceptar.

7.2– Manual de usuario

Para finalizar, se hará una pequeña guía de la aplicación, con la información necesaria de todas sus vistas una vez desplegado el proyecto.
La primera vista que encontramos es la página de landing 7.9, la cual nos explica un poco el proyecto, sus tecnologías y donde podemos contactar.

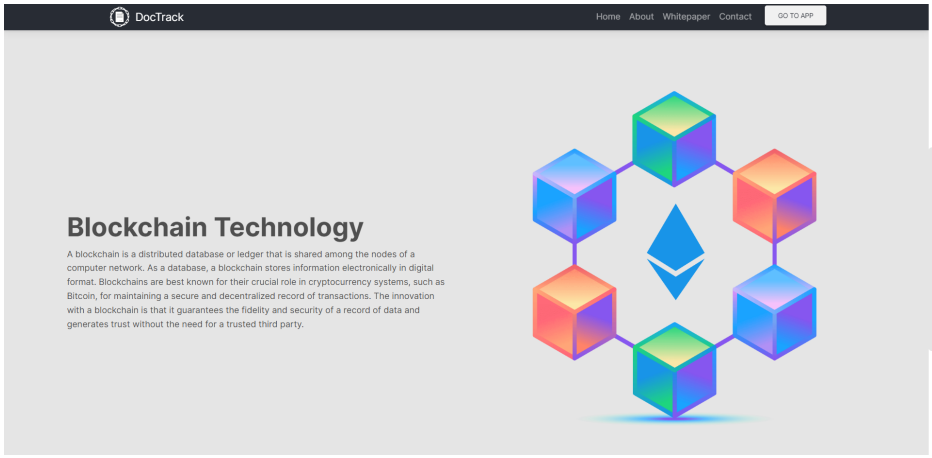

Figura 7.9: Página de landing.

Para avanzar, podemos pulsar en “Go to App“ , arriba a la derecha o en el footer. Esto nos llevará a la bienvenida a nuestra aplicación, ver el la figura 7.10, donde, si estamos registrados nos logueará automáticamente. Si no, deberemos pulsar el botón “Connect wallet“ y aceptar la firma.

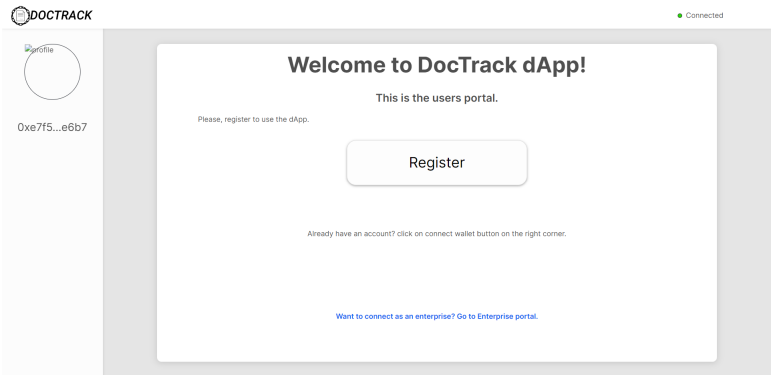

Figura 7.10: Página de bienvenida y registro.

Una vez pulsemos en “Register“ , veremos un formulario de registro, como en la figura 7.11 donde deberemos rellenar los campos deseados.

Como explica en la figura, existen diferentes niveles: 0,1,2. Los cuales indican la cantidad de información que se provee a la empresa, la cual decidirá el nivel mínimo que requiere para que los usuarios interactúen con ella.

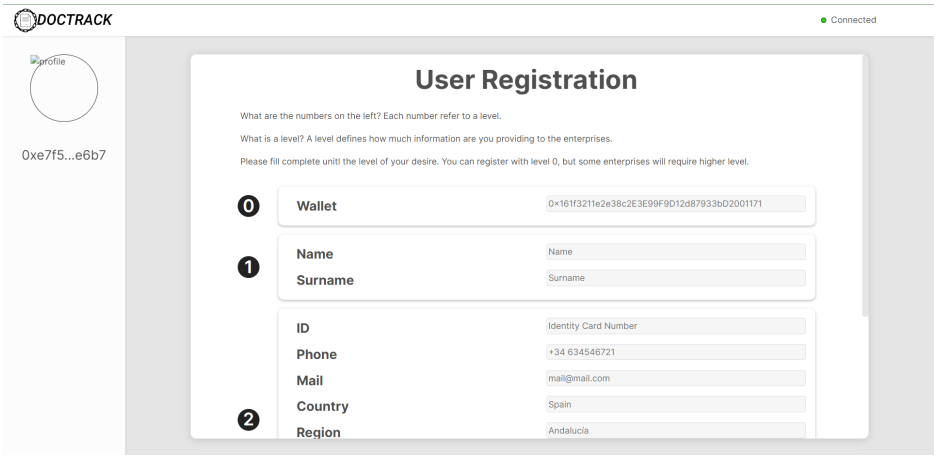

Figura 7.11: Formulario de registro.

Tras rellenar los datos, nos pedirá la firma de una transacción, se muestra un ejemplo en la figura 7.12, para que nuestros datos se almacenen en la blockchain.

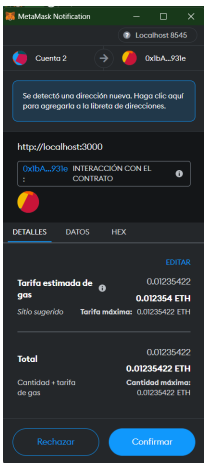

Figura 7.12: Transacción de Metamask.

Una vez firmada la transacción, y procesada por la blockchain, nos redirigirá a la página de inicio, la cual es nuestro dashboard. En esta, podremos ver alguno de los documentos enviados y de las empresas disponibles, aunque no todos.

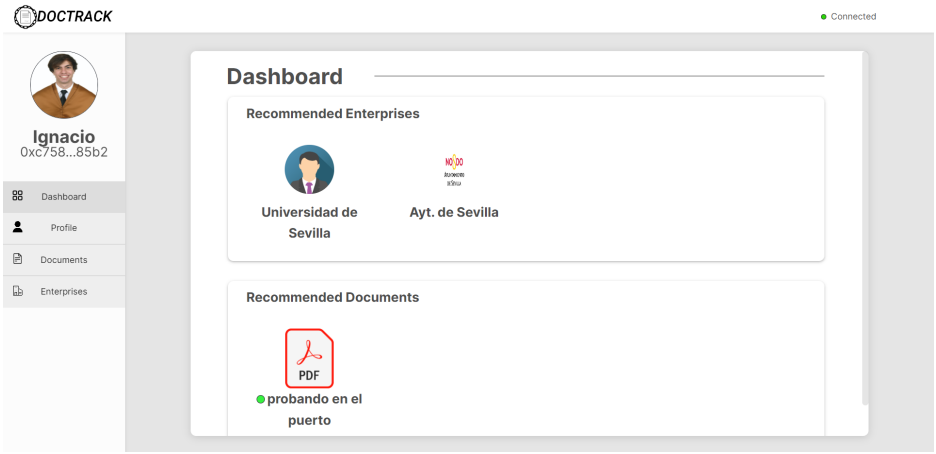

Figura 7.13: Dashboard de usuario.

A nuestra izquierda, en la barra lateral, podremos ver un poco de nuestra información, y las diferentes vistas posibles que tenemos. Si queremos ver más información sobre nosotros, editar nuestra imagen de perfil, o nuestros datos personales, deberemos ir a "Profile" mostrado en la figura 7.14.

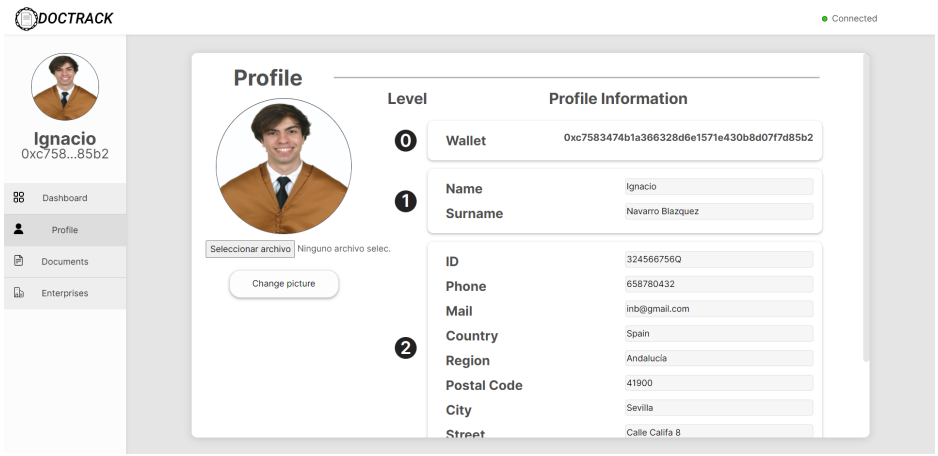

Figura 7.14: Perfil de usuario.

Si deseamos cambiar nuestra foto de perfil, debemos pulsar sobre seleccionar archivo, donde seleccionaremos nuestra foto, y tras ello, pulsaremos sobre "Change Picture". Se nos abrirá una nueva transacción a firmar, tal y como hemos visto en registro, se muestra un ejemplo en la figura 7.12.

También, tendremos la vista de "Documents" en la figura 7.15 la cual obtiene todos los documentos que hemos enviado a las diferentes empresas, con el estado en el que se encuentran.

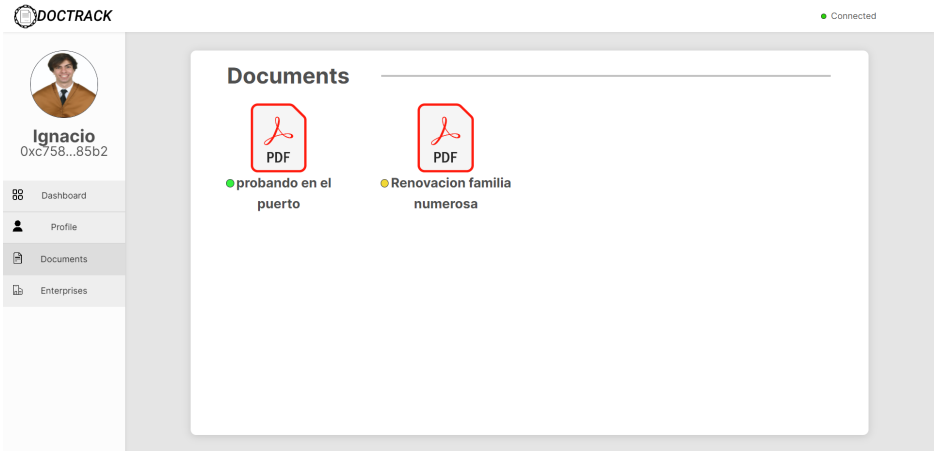

Figura 7.15: Documentos de usuario.

Si pulsamos sobre un documento, veremos la información detallada de este, como se ve en la figura 7.16, con la opción de descargarlo y con la tabla de estados completa, además de con quién ha sido compartido ese documento.

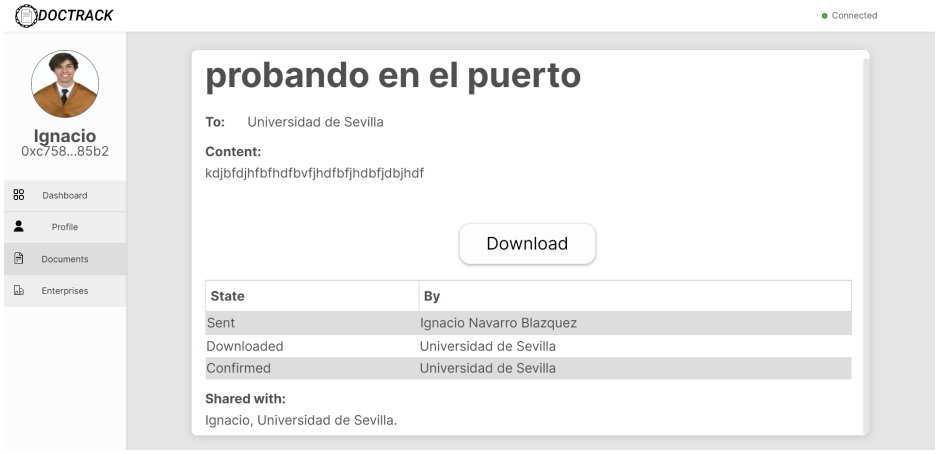

Figura 7.16: Información de documento de usuario.

Por último, tenemos la vista de "Enterprises", la cual nos muestra un listado con todas las empresas registradas en la aplicación, representado en la figura 7.17.

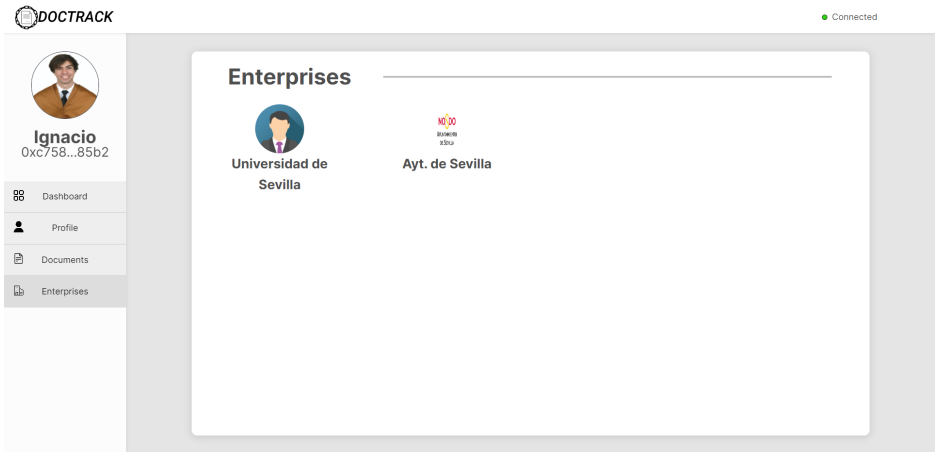

Figura 7.17: Listado de empresas registradas.

Al pulsar sobre cualquiera de ellas, se nos abrirá una pestaña con la información de la empresa como se ve en la figura 7.18, que muestra el nivel mínimo requerido, el email y la cartera de la empresa para cerciorarnos que es la correcta. Al pulsar sobre "more information" nos abre un desplegable con más información sobre esta, como en la figura 7.19.

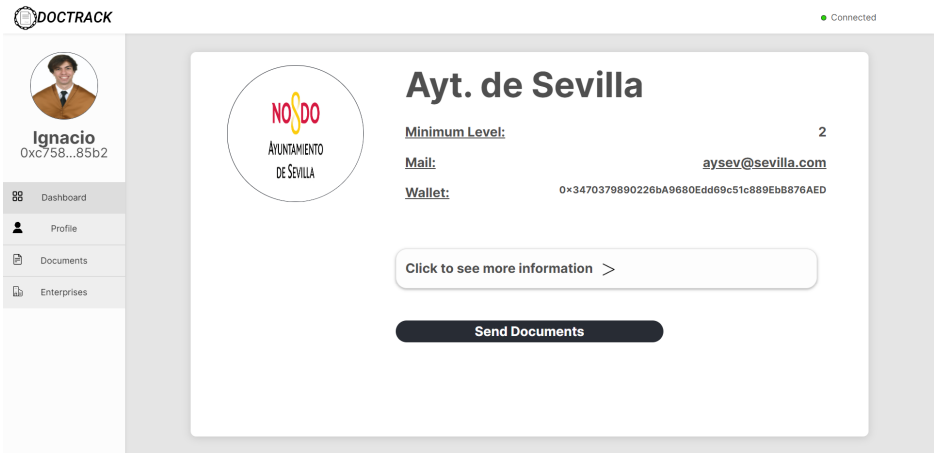

Figura 7.18: Empresa con información no desplegada.

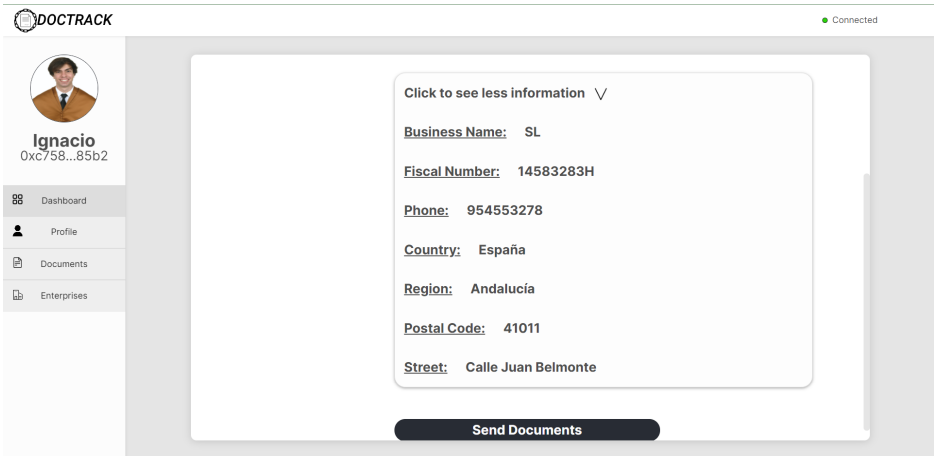

Figura 7.19: Empresa con información desplegada.

Al pulsar sobre “Send Documents“, abriremos una nueva vista, con un formulario a rellenar y un botón para adjuntar archivos, ver en la figura 7.20, y justo debajo, se encuentra el botón para enviar. Al pulsar este botón, se nos abrirá una transacción a firmar, ver ejemplo en la figura 7.12, y tras aceptarla, se enviará nuestro documento y seremos redirigidos a Dashboard de nuevo.

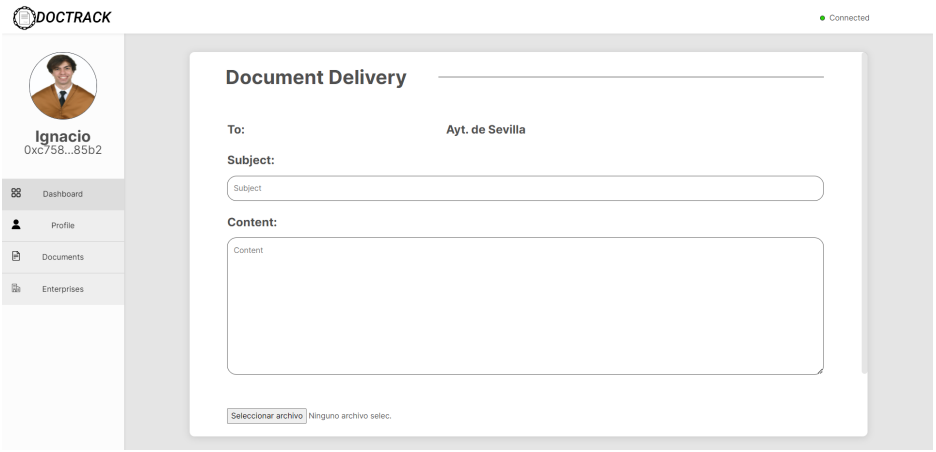

Figura 7.20: Envío de documento.

7.3– Manual de empresa

Procedemos a explicar la parte empresarial. Para entrar a esta vista, podemos hacerlo desde el footer de la página de landing 7.9, o desde la página de bienvenida, ver en la figura 7.10, la cual contiene un hipervínculo al empresarial.

La página de bienvenida no difiere en la de usuario, por lo que es muy parecida a la figura 7.10. Sin embargo, al pulsar sobre "Register" , si hay diferencias como podemos observar en la figura 7.21.

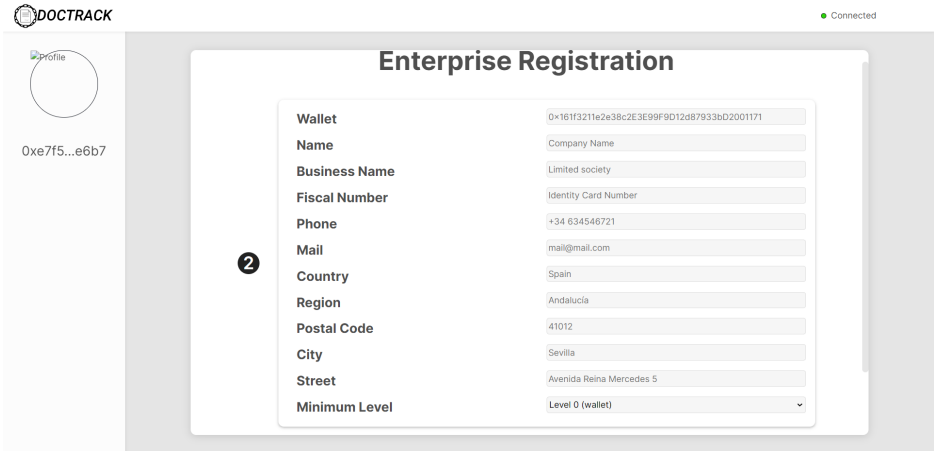

Figura 7.21: Registro de empresa.

Para una empresa, es obligatorio dar toda la información sobre esta, al contrario que un usuario, para evitar ser estafados. Al rellenar todos los datos y seleccionar el nivel mínimo, deberemos firmar dos transacciones como en la figura 7.22, una para registrar los datos de la empresa y otra para registrar el nivel mínimo requerido.

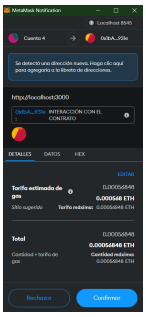

Figura 7.22: Transacción de registro.

Tras el registro, será redirigido como un usuario a su dashboard el cual se ve como en la figura 7.23.

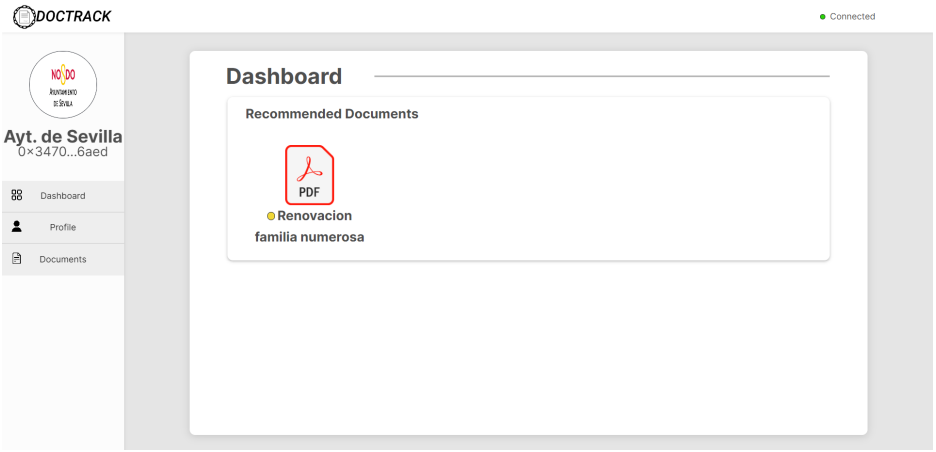

Figura 7.23: Dashboard de empresa.

La pestaña de perfil es muy similar, se puede ver en la figura 7.24, añadiendo el formulario de una empresa y el nivel mínimo, y para actualizar nuestros datos también requeriremos como en usuario firmar las transacciones para que los datos se actualicen en la blockchain.

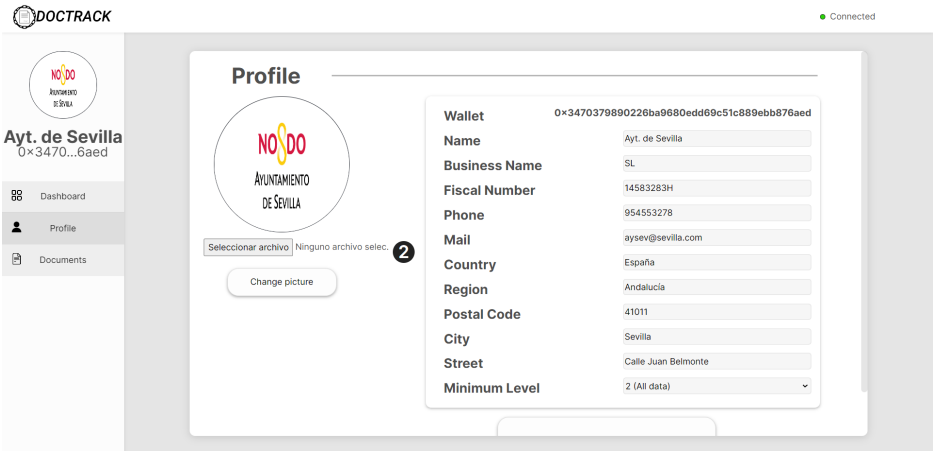

Figura 7.24: Perfil de empresa.

Por último, contaremos con la pestaña de "Documents" en la figura 7.25 que muestra todos los documentos recibidos por la empresa junto con el estado en el que se encuentran.

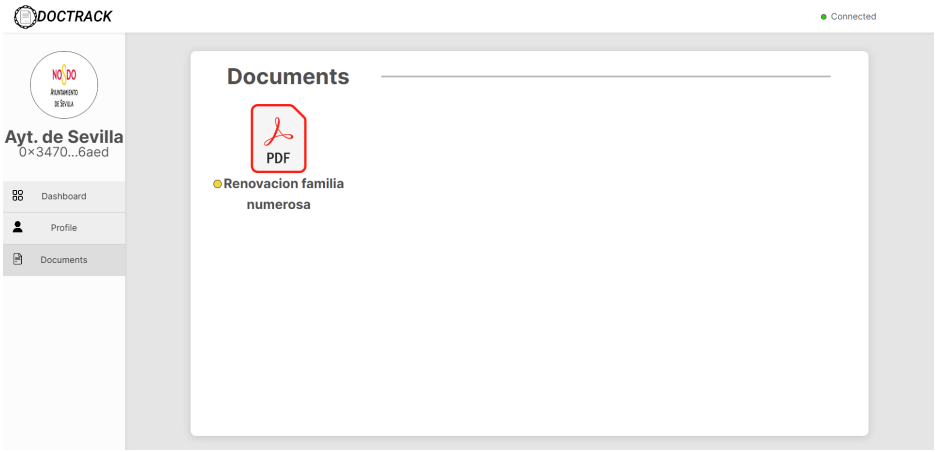

Figura 7.25: Documentos recibidos por la empresa.

Al pulsar sobre uno de ellos, cambiaremos a la vista de la figura 7.26, que contiene la información del documento, junto con su estado, las personas que han pasado por el documento y los botones necesarios para cambiar el estado del documento. Para descargar, confirmar o rechazar el documento, habrá que firmar una transacción como se ve en la figura 7.22.

Figura 7.26: Información documento recibido.

7.4– Manual de documento

En esta sección, detallaremos el código de color que usan los documentos según su estado. Pueden ser los siguientes:

- Amarillo: Representa que un documento ha sido enviado por el usuario y recibido por la empresa 7.27.
- Azul: Representa que un documento ha sido descargado y visto por la empresa que lo recibió 7.28.
- Verde: Representa que la empresa ha aceptado el documento, es decir, está todo correcto 7.29.
- Rojo: Representa que la empresa ha rechazado el documento por que contenía algún fallo 7.30.

Se pueden observar en las siguientes figuras 7.27 , 7.28 , 7.29 y 7.30.

Figura 7.27: Documento enviado y recibido.

Figura 7.28: Documento descargado y visto.

Figura 7.29: Documento confirmado.

Figura 7.30: Documento rechazado.

Bibliografía

[1] Binance Academy. Una introducción al script de bitcoin. https://academy.binance.com/es/articles/an-introduction-to-bitcoin-script, 2020.

[2] Binance. ¿Qué es un ataque del 51 %? https://academy.binance.com/es/articles/what-is-a-51-percent-attack, 2018.

[3] Ignacio Navarro Blázquez. NFTs: Revolution of Ownership. https://www.researchgate.net/publication/362792440_NFTs_The_upcoming_revolution_of_ownership, 2021.

[4] ClickUp. Clickup, One app to replace them all. https://clickup.com/, 2022.

[5] Portal de administración electrónica. La firma electrónica. https://firmaelectronica.gob.es/Home/Ciudadanos/Firma-Electronica.html.

[6] José María Gay de Liébana. Fiebre juvenil por las criptomonedas. https://www.eleconomista.es/opinion-blogs/noticias/11179919/04/21/Fiebre-juvenil-por-las-criptomonedas.html, 2021.

[7] Muhammad Ashraf et al. Bitcoin and Cryptocurrency: Challenges, Opportunities and Future Works. https://www.koreascience.or.kr/article/JAKO202026061031775.page, 2020.

[8] Figma. Figma, the collaborative interface design tool. https://www.figma.com/, 2022.

[9] Ethereum Foundation. Ethereum Foundation. https://ethereum.foundation/, 2014.

[10] Ethereum Foundation. Especificación de application binary interface. https://solidity-es.readthedocs.io/es/latest/abi-spec.html, 2017.

[11] Ethereum Foundation. Ethereum. https://ethereum.org/es/, 2017.

[12] Ethereum Foundation. Solidity. https://solidity-es.readthedocs.io/es/latest/, 2017.

[13] Glassdoor. Búsqueda de sueldos y remuneración en reino de españa. https://www.glassdoor.es/Sueldos/index.html, 2021.

[14] IEEE. Recommended practice for software requirements specifications. https://standards.ieee.org/ieee/830/1222/, 2009.

[15] ConsenSys Software Inc. Ganache, one click blockchain. https://trufflesuite.com/ganache/, 2016.

[16] ConsenSys Software Inc. Metamask, la billetera de criptomonedas para defi, web3 dapps y nfts. https://metamask.io/about/, 2022.

[17] Infura. Infura, the Gateway to Blockchain Development. https://infura.io/, 2022.

[18] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[19] Antoni Paszke. Layer 2. https://academy.binance.com/en/glossary/layer-2, 2022.

[20] Purwono Purwono, Alfian Maárif, Wahyu Rahmaniar, Qazi Mazhar Ul Haq, Dimas Herjuno, and Muchammad Naseer. Blockchain technology. 8:199–205, 07 2022.

[21] SCRUM. What is scrum? https://www.scrum.org/resources/what-is-scrum, 2021.

[22] Corwin Smith. MÁQUINA VIRTUAL DE ETHEREUM (EVM). https://ethereum.org/es/developers/docs/evm/, 2022.

[23] Wikipedia. Know your customer. https://es.wikipedia.org/wiki/Conozca_a_su_cliente, 2022.

[24] Wikipedia. Peer-to-peer. https://es.wikipedia.org/wiki/Peer-to-peer, 2022.

[25] Wikipedia. Web3. https://en.wikipedia.org/wiki/Web3, 2022.

[26] Zendesk. Estrategias para satisfacer al cliente. https://www.zendesk.com.mx/blog/estrategias-para-satisfacer-clientes/.