Plataforma blockchain para visualizar la reputación de empresas sobre casos de discriminación

Departamento de Ingeniería del Software e Inteligencia Artificial

UNIVERSIDAD COMPLUTENSE DE MADRID

Facultad de Informática

TRABAJO DE FIN DE GRADO

Ana Belén Duarte León
Ángeles Plaza Gutiérrez
Alejandro Ramírez Rodríguez
David Seijas Pérez
Jorge del Valle Vázquez
Javier Verde Marín

Director/es: Samer Hassan, Silvia Semenzin

Doble Grado en Matemáticas e Ingeniería Informática y Grado en Ingeniería Informática

Curso académico 2021-2022

Agradecimientos:

Quisiéramos dejar constancia de nuestra gratitud para con nuestros tutores, el Investigador y Profesor Samer Hassan Collado y la Doctora Silvia Semenzin, por guiarnos a lo largo de este proyecto y por todo su tiempo y atención dedicada en solventar todas y cada una de nuestras incidencias.

A Don Carlos Alcaide Pastrana por prestarnos su apoyo gracias al programa de co-tutorización y apoyo a TFGs de blockchain de Telefónica.

A nuestros amigos y compañeros de clase, que han sido un fuerte apoyo durante nuestra estancia en la Universidad.

A lo profesores, docentes e investigadores de la Universidad por su dedicación e interés a la hora de trasmitir los conocimientos requeridos.

Y, por último, a nuestras familias y amigos que nos han brindado siempre su apoyo desinteresado en el transcurso de nuestros estudios.

Resumen

En el presente proyecto se ha desarrollado una **DApp** (Decentralized App) de arquitectura híbrida cuya principal funcionalidad es la de **denunciar** casos de discriminación en el entorno laboral, así como observar la **reputación** de las distintas empresas.

El núcleo de la aplicación es un Sistema **Descentralizado**, basado en la tecnología **Blockchain**. Gracias a sus características, aporta a los empleados la capacidad de dar a conocer su experiencia en una empresa impidiendo a esta manipular, modificar o directamente eliminar la denuncia. Así conseguimos que nuestra aplicación ofrezca toda la confianza necesaria a las víctimas.

Con el fin de ofrecer anonimato y mostrar métricas interactivas en un tiempo de respuesta eficiente se ha construido un Sistema **Centralizado** escalable horizontalmente gracias a la tecnología de contenedores de **Docker**. Este sistema está diseñado de forma que los datos almacenados no puedan vulnerar el derecho a la privacidad y contribuya a garantizar la transparencia que la aplicación desea promover.

La UI (interfaz de usuario) está diseñada con el objetivo de ser lo más intuitiva posible, favoreciendo al usuario la navegación a través de las distintas secciones que componen la aplicación web. No requiere poseer ningún conocimiento previo sobre sistemas descentralizados o blockchain.

Palabras clave: Descentralizado, Blockchain, Ethereum, Smart Contract, IPFS, Docker, discriminación, denuncia, reputación.

Abstract

In this project we have developed a **DApp** (Decentralized App) with a hybrid architecture whose main feature is to **condemn** cases of discrimination in the workplace, as well as to check the **reputation** of different companies.

The core of the application is a **Decentralized** System, based on **Blockchain** technology. Thanks to its properties, it provides the employees with the ability to share their experience at a company, preventing the company from manipulating, modifying or even deleting the complaint. In this way, we ensure that our application offers all the necessary confidence to the victims.

In order to provide anonymity and show interactive metrics in an efficient response time, we have built a horizontally scalable **Centralized** System by using **Docker** container technology. This system is designed in order to ensure that the stored data could not breach the right to privacy and helps guarantee the desired transparency which the application desires to promote.

The UI (User Interface) is designed to be as user-friendly as possible, favoring the navigation for the user through the different sections that constitute the web app. It doesn't require any previous knowledge of blockchain or decentralized systems .

Keywords: Decentralized, Blockchain, Ethereum, smart contract, IPFS, Docker, discrimination, complaint, reputation.

Índice

Índice de cuadros

Índice de figuras

1. Introducción

1.1. Motivación

La sociedad se encuentra en una etapa en la que dar voz a los problemas y ser escuchado es de gran importancia. La tecnología actual permite compartir experiencias de forma rápida y sencilla, llegando a la máxima cantidad de personas posibles. A pesar de los muchos avances tecnológicos, aun carecemos de plataformas centradas en fomentar el bienestar de los ciudadanos, por ejemplo desde el punto de vista laboral. Los datos que conocemos de las empresas casi nunca son generados por parte de los propios trabajadores. Por ello, nuestra plataforma nace con el objetivo de revertir la situación y dar voz a los empleados. A partir de estas reflexiones, surge este canal de visibilización de la discriminación en entornos de trabajo.

Por desgracia, a menudo se dan situaciones repulsivas y espantosas en el ámbito laboral. Algunos de estos casos acaban ante un juez, aunque pasan desapercibidos para la sociedad. Dar voz a las víctimas para que sientan que denunciar tiene un impacto en la empresa y en los agresores, es un aspecto clave en la búsqueda de la transparencia y la justicia. Es por ello que surge la búsqueda de un canal de transmisión cuya principal motivación sea la dar visibilidad a estos problemas.

Actualmente las empresas cuentan con diversos recursos para tratar que las denuncias no salgan a la luz. Sin embargo, los sistemas descentralizados suponen un buen antídoto, pues escapan de cualquier influencia. Entre todas ellas destaca Blockchain, piedra angular de este proyecto. Mediante el uso de esta tecnología, se aporta persistencia al sistema de denuncias imposibilitando los intentos de eliminación u ocultación; además de transparencia y confianza por parte de los denunciantes debido a su carácter público.

Asimismo, otro de los motivos por los que no se suele denunciar en el ámbito laboral es el miedo a las posibles consecuencias, como por ejemplo la pérdida de empleo. Por lo tanto, este proyecto debe generar un ecosistema que incentive a las víctimas a contar sus historias. Es primordial que los denunciantes se sientan protegidos, ocultando al máximo posible su identidad.

1.2. Objetivos

Al tratarse de una aplicación que combina innovación tecnológica con una importante labor social, los objetivos del proyecto se centran en ambas secciones. Las dos ramas son igual de importantes y a su vez se complementan, ya que el uso de la tecnología está enfocado en obtener la mejor experiencia para el usuario. Los objetivos se muestran en dos listas, ordenadas en función de su importancia.

Sociológicos

- Ofrecer a las víctimas la certeza de que su denuncia sí va a tener un impacto en la empresa. A su vez, es esencial asegurar que ninguna compañía puede interferir en su denuncia.
- Mantener el anonimato del denunciante.
- Promover el acto de denunciar discriminaciones en el ámbito laboral.
- Establecer un sistema de reputación robusto y fiel a la realidad, para que los usuarios analicen la situación de cada empresa.

Tecnológicos

- Desarrollar una aplicación que consiga satisfacer todos los objetivos sociológicos expuestos.
- Aprovechar todas las ventajas que ofrece Blockchain, y en particular Ethereum, para generar un ambiente de confianza y transparencia.
- Confeccionar una interfaz sencilla, intuitiva, que recoja todas las funcionalidades y que integre con naturalidad todas las tecnologías implementadas.
- Desarrollar un Back-End eficiente y escalable, que permita la conexión de múltiples usuarios instantáneamente a la aplicación.

1.3. Estructura del documento

Esta memoria explica el proceso de desarrollo de la plataforma a lo largo de todas sus fases, desde la investigación hasta la implementación de la versión definitiva y su evaluación.

En los capítulos 3 introducimos la planificación que ha seguido el equipo, así como la organización interna y la repartición del trabajo. En el 4 se especifican todas las tecnologías empleadas, diferenciando en que capa se han empleado.

Por su parte, los capítulos 5 y 6 son los capítulos principales, en los que se desarrolla la arquitectura e implementación respectivamente. Ambos capítulos siguen la misma estructura interna: división en Front-End y Back-End, y dentro del Back en Sistema Descentralizado y Sistema Centralizado. El capítulo de implementación a su vez está fraccionado según la versión: inicial, una especie de prototipo funcional; y la definitiva, evaluada por usuarios y presentada como proyecto final.

El capítulo 7 narra las tareas realizadas por cada miembro del equipo. En el 8, se describe todo el proceso de la evaluación realizada con usuarios. Al final del propio capítulo se recopilan las conclusiones extraídas de las pruebas.

Por último, tenemos un capítulo que trata los siguientes apartados: límites que existen en el proyecto, especialmente por la tecnología Blockchain; recomendaciones para trabajo futuro; y la conclusión del proyecto, basada en el aprendizaje y el resultado obtenido.

2. Introduction

2.1. Motivation

The society is at a stage in which giving voice to social problems and being heard is extremely important. Current technology allows you to share experiences quickly and easily, reaching the maximum number of people possible. Despite the many technological advances, we still lack platforms focused on promoting the well-being of citizens. One of the areas in which this is utterly important is the relation between workers and companies. The data that we know about companies is almost never generated by the workers themselves. For this reason, our platform was born with the aim of reversing the situation and giving employees a voice. From these thughts, arises this channel of visibility of discrimination in work environments.

Unfortunately, repulsive and frightening situations often occur in the workplace. Some of these cases end up in a jury, although they go unnoticed by society. Giving victims a voice so that they feel that reporting has an impact on the company and on the aggressors is a key aspect in the search for transparency and justice. That is why the search for a transmission channel arises whose main motivation is to give visibility to these problems.

Companies currently have various resources to try to ensure that complaints do not come to light. However, decentralized systems are a good antidote, as they escape any influence. Among all of them, Blockchain stands out, the cornerstone of this project. Through the use of this technology, persistence is provided to the reporting system, making it impossible to eliminate or conceal attempts; in addition to transparency and trust on the part of the complainants due to its public nature.

Likewise, another of the reasons why it is not usually reported in the workplace is the fear of possible consequences, such as loss of employment. Therefore, this project must generate an ecosystem that encourages victims to tell their stories. It is essential that whistleblowers feel protected, hiding their identity as much as possible.

2.2. Objetives

Since it is an app combining technological innovation and important social work, the project's objectives are focused on both aspects. They are equally important and also complement each oder, as the use of technology i focused on getting the very best experience for the user. The objectives are shown in two lists, ordered by importance.

Sociological

- Offer victims the confidence that their complaint will indeed have an impact on the company. It is essential to ensure that no company will interfere with their report.
- Keep the anonymity of the complainant.
- Encourage the act of denouncing workplace discrimination.
- Build a solid and reliable reputation system so users can review the company's situation.

Technological

- Develop an application that succeeds in fulfilling all the sociological objectives stated above.
- Take every advantage of the Blockchain, and in particular Ethereum, to create an atmosphere of trust and transparency.
- Design a simple, user-friendly interface that includes all the functionalities and naturally integrates all the technologies used.
- Create an efficient and scalable Back-End, allowing multiple users to instantly connect to the app.

2.3. Document structure

This memory explains the development process of the platform throughout all its stages, from investigation to the implementation of the final version and its evaluation.

In chapter 3 we present the team planning, as well as the internal organisation and the work distribution. Chapter 4 describes in detail all the technologies used, specifying in which layers they have been employed.

Chapters 5 and 6 are the core chapters, in which architecture and implementation are developed respectively. Both chapters share the same internal structure: division into Front-End and Back-End, and within the Back-End into Decentralised System and Centralised System. The implementation chapter is divided according to the versions: initial, a kind of functional prototype; and the final version, which is evaluated by users and is presented as the final project.

Chapter 7 describes the task carried out by each member of the team. Chapter 8 covers the whole process of the evaluation carried out with users. The conclusions drawn from the tests are presented at the end of the same chapter.

Finally, there is a chapter that deals with the limits that exist in the project, especially due to the Blockchain technology; the recommendations for future work; and the project's conclusion, which is based on the learning and the result achieved.

3. Estado del Arte

Detallamos a continuación, las investigaciones y publicaciones realizadas para el desarrollo de este proyecto. Se presentarán en orden cronológico y de forma ordenada tal y como lo realizó el equipo, pues se partía desde el completo desconocimiento de la tecnología Blockchain y de la red Ethereum. Además de la investigación tecnológica, se realizó en paralelo una investigación de carácter más social para sustentar la parte sociológica de la plataforma. El estudio de esta materia realizado será detallado igualmente en esta sección.

3.1. Sistemas descentralizados

Para que una red se califique como descentralizada [1], puede serlo desde tres puntos de vista, arquitectónico, lógico y político. La descentralización arquitectónica hace referencia al número de ordenadores o nodos que componen el sistema y cuantos de ellos puede tolerar el sistema de tener una caída simultanea. La descentralización política se pregunta cuántos son los individuos u organizaciones que controlan los ordenadores de los que se compone el sistema. La descentralización desde un punto de vista lógico indica si la estructura o interfaz del sistema se conforma de un único gran objeto o bien de un conjunto de objetos. Por tanto, lo verdaderamente importante no es calificar globalmente un proyecto o sistema como descentralizado, sino valorar la descentralización en los distintos elementos.

Las cadenas de bloques, que posteriormente trataremos son:

- Políticamente descentralizadas, pues nadie ejerce un dominio o control.
- Arquitectónicamente descentralizadas, porque carece de una infraestructura central y sería imposible una caída total del servicio si se diera una caída parcial de la red.
- Lógicamente centralizada, hay un estado general común para todos los nodos, por lo que el sistema funciona como unidad.

Las ventajas de un sistema descentralizado son la resistencia a fallos, ataques y colisiones. Son resistentes a fallos al seguir funcionando aunque haya fallos en partes de la red, puesto que se apoyan en el resto de nodos siguen operativos. La resistencia a los ataques radica en el coste de efectuar los ataques comparado con un sistema que posea un punto central, siempre que existan suficientes nodos conectados. Los ataques de colisión son aquellos en los que participantes del sistema actúan de forma que se ven beneficiados a costa de otros.

La importancia de la descentralización recae no solo en resistir los ataque antes mencionados, además su puede considerar que facilitan la creación de sistemas sin censura y permiten que la información no quede en manos de unos pocos dominadores que decidan a a sus anchas que hacer con ella.

3.2. Introducción a Bitcoin y conceptos existentes

El concepto de moneda digital descentralizada se ha nombrado durante décadas[2]. Los protocolos anónimos de dinero electrónico de los años 80 y 90 se basaron principalmente en un primitivo criptográfico conocido como Chaumian Blinding. Chaumian Blinding proporcionó estas nuevas mo-

nedas con altos grados de privacidad, pero los protocolos subyacentes no consiguieron obtener gran tracción o bienvenida debido a que dependían de un intermediario centralizado. En 1988 el b-money de Wei Dai surge como la primera propuesta de creación de dinero a través de la solución de rompecabezas computacionales, así como del consenso descentralizado. Sin embargo, esta primera propuesta no contenía demasiados detalles de cómo implementar este consenso descentralizado. En 2005, Hal Finney introduce el concepto de "prueba de trabajoün sistema que usaba ideas de b-money junto con rompecabezas de dificultad computacional para crear el concepto de criptomoneda, aunque una vez más, no alcanzó el ideal de criptomoneda descentralizada puesto que dependía de un cómputo de confianza como Back-End.

En octubre de 2008, Satoshi Nakamoto publicó una propuesta donde plantea crear un protocolo llamado Bitcoin sobre una red P2P con la finalidad de transferir valor, representado por una criptomoneda llamada Bitcoin, de un punto a otro de la red, sin necesidad de interactuar con terceras partes, como pueden ser bancos, plataformas de internet u otras instituciones. Así, este protocolo propone una solución al problema de la confianza en los intermediarios, al garantizar a los partícipes de la red que pueden confiar en los resultados producidos por la misma sin confiar los unos en los otros o siquiera conocerse.

Bitcoin combina las bases establecidas para manejar la posesión mediante una llave criptográfica pública con un algoritmo de consenso para mantener el registro de quién posee monedas. Desde un punto de vista más técnico, el Bitcoin puede ser considerado un sistema de transición de estado, donde hay un estado que contiene la información de la propiedad de todos los Bitcoins existentes y una función de transición de estado que toma un estado y una transacción y produce como salida un nuevo estado que es el resultado. El protocolo de Bitcoin opera sobre redes P2P, formadas por nodos iguales entre sí en las que no hay servidores o clientes fijos. Además, la seguridad de la red y el protocolo, desde el almacenamiento de datos hasta el traspaso de información entre nodos se apoya por completo en métodos criptográficos.

La idea de tomar la subyacente idea de la cadena de bloques y aplicarla a otros conceptos también tiene una larga historia. En 2005, Nick Szabo presenta el concepto de títulos de propiedad seguros con autorización del propietario, un documento que describe como "nuevos avances en la tecnología de bases de datos replicadas"permitirán un sistema basado en cadenas de bloques para almacenar un registro de quién posee qué tierra, creando un marco elaborado que incluye conceptos como homestanding, posesión adversa y el impuesto Gregoriano a la tierra. Después de 2009, una vez que se desarrolló el consenso descentralizado de Bitcoin, rápidamente comenzaron a surgir varias aplicaciones alternativas.

Para el asegurar y proteger intercambio monetario en una red P2P, Bitcoin se apoya fuertemente en la Blockchain, una tecnología que presenta soluciones a grandes problemas de este tipo de redes como es el doble gasto.

La blockchain es como dicen Don y Alex Tapscott en su libro Blockchain Revolution: “un libro de contabilidad digital incorruptible de transacciones económicas que se puede programar para registrar no solo transacciones financieras, sino prácticamente todo lo que tiene valor”. Pero ¿cómo funciona?

A medida que se produce una transacción, se registra como un "bloque"de datos. Estas transacciones muestran el movimiento de un activo, el cual puede ser tangible o intangible. Cada bloque

está conectado al bloque anterior y al bloque posterior formando una cadena de datos a medida que un activo se mueve de un lugar a otro o cambia de dueño. Los bloques confirman tanto el tiempo exacto como la secuencia de las transacciones y se unen de forma segura para evitar que se alteren o se inserten entre dos bloques existentes. Cada bloque adicional refuerza la verificación del bloque anterior y, por lo tanto, de todo el blockchain. Esto hace que dicha cadena sea a prueba de manipulaciones, lo que constituye la ventaja principal de la inalterabilidad. Esto evita que alguien malintencionado modifique la cadena y crea un libro mayor distribuido de transacciones en la que los nodos de la red pueden confiar.

Dentro de la red cada nodo participante posee una copia de la última versión de la cadena de bloques. Al no requerirse intermediarios, la red valida sus propias transacciones con un algoritmo de consenso llamado "prueba de trabajo" cuyas ideas surgieron años antes, como se ha comentado. Para seleccionar al próximo nodo validador se propone una prueba matemática que deben resolver todos y cada uno de los nodos utilizando su poder de cómputo para encontrar la solución por fuerza bruta.

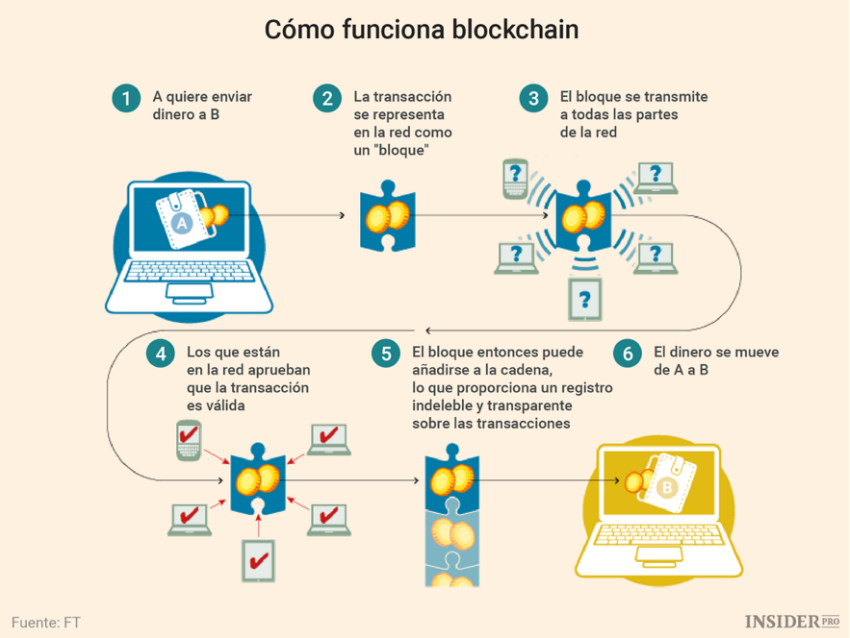


Figura 1: Funcionamiento de blockchain
Tomado de [3]

3.3. Ethereum y Blockchain

Desde un punto de vista de la informática, Ethereum es una máquina de estados abstracta pero prácticamente ilimitada, que consiste en un estado único a nivel mundial y una máquina virtual que aplica los cambios a ese estado. Por ello, se le conoce como "el ordenador mundial".

Desde una perspectiva más práctica, Ethereum es una infraestructura informática de código abierto y descentralizada a nivel mundial que ejecuta programas denominados contratos inteligentes. Usa una cadena de bloques para sincronizar y almacenar los cambios de estado del sistema, junto

con una criptomoneda llamada ether para medir y limitar los costes de los recursos de ejecución.

Ethereum permite a los desarrolladores crear potentes aplicaciones descentralizadas con funciones económicas incorporadas. A la vez que proporciona alta disponibilidad, auditabilidad, transparencia y neutralidad, también reduce o mitiga la censura y ciertos riesgos de contrapartida.

En general, existe un conocimiento mucho más extendido de qué es Bitcoin y cómo funciona y esto puede llevar a asunciones sobre Ethereum que no son correctas. El objetivo principal de Ethereum no es convertirse en una red de pago de una moneda digital. Mientras que la moneda digital ether es integral y necesaria para el funcionamiento de Ethereum, el ether está pensado como una moneda de utilidad para pagar por el uso de la plataforma Ethereum, como se estableció antes, por los costes de ejecución.

3.3.1. Smart Contracts

Un Smart Contract o Contrato Inteligente es un programa informático que implica un contrato auto ejecutable que engloba los términos de acuerdo entre comprador y vendedor, escrito en código y guardado en una red Blockchain distribuida y descentralizada.

Existen dos tipos de plataformas blockchain, Permissioned y Permissionless (o Public). Los requisitos y expectativas varían dependiendo de la plataforma, mientras que en una pública cualquiera puede desplegar contratos, con un coste asociado que previene el spam, en la otra, el acceso es exclusivo para usuarios con permiso. De acuerdo a lo anterior, cada tipo es más apropiado para determinados contratos inteligentes[4].

Encontramos ejemplos de aplicación de contratos inteligentes en:

- Asistencia sanitaria, guardando historiales médicos[5].
- IoT, para gestión de identidades como uPort [6] (proteger privacidad de usuarios tras pérdida de dispositivos) o Sovrin[7], Watson IOT Platform de IBM[8], para proporcionar una red empresarial de confianza, o Chain of Things[9] que fusiona blockchain con IoT para lograr seguridad, fiabilidad e interoperabilidad.
- Banca, para reducir gastos tanto de clientes como del propio banco, tenemos a Stellar[10] que se construye sobre el proyecto Hyperledger Fabric[11] que está impulsado principalmente por la Linux Foundation.
- Cadenas de suministro, para garantizar transparencia, seguridad y visibilidad[12].
- Votación, para garantizar la equidad y la transparencia de las elecciones, puesto en práctica por un partido político danés[13][14].
- Seguros, aumentando transparencia a los consumidores y reduciendo el coste administrativo global para las aseguradoras[15].

La lista de posibles aplicaciones expuestas representa parte del conjunto total de posibilidades, que crece cada día y que siguen en proceso de investigación, pues todavía no se ha hallado solución a múltiples problemas, siendo uno de los más importantes la garantía de privacidad, necesaria en los historiales médicos o dispositivos IOT mencionados.

3.3.2. Tokens

Los tokens son unidades con valor asociado, que una organización cualquiera crea para modelar su negocio, dar poder a los usuarios para interactuar con los productos o facilita la distribución de beneficios entre accionistas. Tienen múltiples aplicaciones, desde representar activos monetarios, hasta acciones, smart properties, sistemas de reputación, etc. Estos sistemas de tokens se implementan de forma muy sencilla en Ethereum siguiendo una simple lógica en un contrato. La clave es que todo sistema de token o currency es fundamentalmente una base de datos con una única operación de transferencia entre dos extremos, que resta una cantidad de tokens de un extremo, siempre que disponga de ellos, para sumarlos a los tokens del otro extremo, todo ello con la aprobación del primero.

En esencia, es similar a una función de transición de estado de sistemas bancarios, y solo necesita de otras pocas funcionalidades para completar la lógica del sistema. Una diferencia respecto a las meta-monedas de Bitcoin es que los sistemas de tokens en Ethereum que actúan como submonedas tienen la capacidad de pagar tasas de transacción directamente. Para ello, el contrato mantiene un saldo de ether que sirve para reembolsar al emisor el coste de las tasas. Este saldo se nutre recogiendo la submoneda interna como comisión y posteriormente venderla en un ciclo continuo.

3.3.3. DApps

DApp hace referencia a las Aplicaciones Descentralizadas. Se denomina así a cualquier plataforma que facilita la interacción directa entre usuarios apoyándose en la Blockchain para cerrar acuerdos, vía web o aplicación sin necesidad de una entidad central que gestione el servicio. El control que ejerciera un agente central de cualquier aplicación centralizada se distribuye entre todos los miembros gracias a una cadena de bloques. En esencia, cada usuario de la DApp es un nodo de la red descentralizada que toma nota de cada movimiento realizado en la plataforma. El propio sistema verifica la validez de los movimientos ejecutando contratos inteligentes cuando se den las condiciones estipuladas en su código. A día de hoy ha cerca de 4000 DApps que quedan recogidas en el portal The State of The ĐApps [**stateofDApps**].

La innovacion reside en desarrollar soluciones a las complicaciones de la vida cotidiana o mundo real y garantizar una libertad digital frente a las aplicaciones tradicionales, con mayor flexibilidad y transparencia. Son varias las plataformas sobre las que se puede desarrollar una DApp, tanto con Web3 sobre navegador, como de forma nativa sobre iOS/Android. Entre ellas destacan Ethereum, EOS, Tron y Binance Chain, aunque es Ethereum la considerada más adecuada debido a Solidity, el lenguaje Turing-Completo que ofrece para la creación de los Smart Contracts empleando la EVM (Ethereum Virtual Machine).

A grandes rasgos, son aplicaciones informáticas, mayoritariamente Open Source, que se comunican con la Blockchain y utilizan contratos inteligentes para gestionar las interacciones de los usuarios. Se diferencian en el Back-End respecto a una web tradicional en que no solo utilizan una API para conectarse a la base de datos, sino que interactúan con una Blockchain P2P descentralizada como Ethereum con Smart Contracts. Por otro lado, el Front-End puede desarrollarse desde cualquier interfaz imaginable.

Por ahora la experiencia del usuario promedio con las DApps puede ser poco amigable. Uno

de los problemas más destacables de estas DApps es el conocido como 'bounce rate' que alude a la cantidad de usuarios que cierran la aplicación por las trabas que por ahora se encuentran al usarlas. La capacidad de transacciones que puede sostener la red de Ethereum todavía no permite dar el mismo servicio, en algunos sectores, que otras aplicaciones tradicionales. Hasta el sector DeFi en auge maneja cantidades muy inferiores a los mercados tradicionales. Algunas plataformas como Solana buscan ofrecer soluciones a estos problemas, aumentando el número de transacciones y reduciendo su coste.[16]

Del mismo modo, la forma habitual de interactuar con DApps en la web es Metamask, que establece un puente entre web2 (internet comun) y web3 (blockchain). Su uso requiere un minimo conocimiento sobre carteras de Ethereum, y necesita poder permitirse comprar ETH. Todo esto queda fuera del alcance para la gran mayoría de los usuarios de internet. Surgen nuevas alternativas que buscan incorporaciones mas sencillas sin necesitar de las 'seed phrases' ni extensiones de navegador como Metamask. Authereum o TrustWallet(mobile) por ejemplo, permiten interactuar con las DApps directamente con la cartera sin necesidad de extensiones.

3.3.4. DAOs

La tecnología Blockchain da origen a un nuevo paradigma de autogobernanza en sistemas descentralizados carentes de autoridad central conocidos como DAO. En esencia, una DAO es una organización en la que la interacción entre miembros se da por una aplicación Blockchain controlada por una serie de reglas descritas en su código. De forma autónoma organizan contrataciones, servicios, ganar dinero, poseer propiedades (dentro de la Blockchain),etc.

De acuerdo al artículo Decentralized Autonomous Organization[17] una DAO es un sistema basado en blockchain que permite a las personas coordinarse y auto-gobernarse a través de un conjunto de reglas auto-ejecutables desplegadas en una blockchain pública, y cuya gobernabilidad es descentralizada (es decir, independiente de un control central). Otra definición, menos formal, en palabras de Luis Cuende, "Una DAO es una entidad nativa de Internet sin gestión centralizada que se regula por un conjunto de reglas automáticamente ejecutables en una blockchain pública, y cuyo objetivo es tomar vida propia e incentivar a las personas para lograr una misión común compartida" [18].

Colony [19] sirve de ejemplo de DAO que establece un sistema de reputación meritocrático, es decir, recompensa con reputación , o ether en algún otro caso, por completar una tarea. Divide la organización en dominios que representan equipos, departamentos, proyectos... que tienen una reputación asociada que disminuye con el tiempo para restar valor a las tareas realizadas en el pasado.

3.3.5. IPFS

El sistema de archivos interplanetario (IPFS)[20] es un algoritmo que implica una red peer to peer para almacenar y compartir archivos de forma distribuida, que busca una web descentralizada más rápida y eficiente. El mecanismo utiliza el direccionamiento basado en el contenido, es decir, cada archivo se divide en chunks de menor tamaño que se convierten en hash (basado en el contenido) y se almacenan en una red descentralizada. Para acceder a los archivos solo son necesarias las direcciones hash, que identifican de manera única los archivos. IPFS puede incorporarse con blockchain para

proporcionar características como la inmutabilidad, la alta fiabilidad y el rendimiento. Además, tiene un buen rendimiento y mayor disponibilidad. Principalmente se emplea en plataformas de almacenamiento para compartir datos.

IPFS está dirigido al contenido, lo que significa que se puede acceder a los datos rápidamente por fuentes cercanas al nodo y guarda una copia de la dirección del contenido. IPFS utiliza una tabla hash distribuida (DHT) para almacenar los datos. Se realizan peticiones a los nodos para buscar cualquier contenido. Una vez que se encuentra el contenido se vuelve a solicitar la localización del peer para obtener el contenido requerido.

3.4. Plataformas de denuncia

El activismo de datos o Data activism [21] es un fenómeno empírico de rápido crecimiento que se encuentra en la intersección de las dimensiones social y tecnológica de la acción humana. Es la forma más reciente de activismo mediático y una herramienta heurística capaz de devolver la agencia democrática al análisis de cómo el Big Data afecta a la sociedad contemporánea. Se distinguen dos formas de activismo, proactiva y reactiva. En nuestro caso, atendiendo a la visión proactiva, busca hacer uso de toda posibilidad de defensa y campaña para fomentar el cambio social, como podrían ser las nuevas tecnologías y la gran disponibilidad de datos. Por ejemplo, Occupy Data es una movilización que pretende apoyar iniciativas mediante la recopilación, el análisis y la visualización de datos. A su vez, anima a los activistas a potenciar sus esfuerzos de defensa representando hechos de forma que sean al mismo tiempo "moralmente convincentes, emocionalmente poderosos y racionalmente innegables".

A día de hoy existen múltiples plataformas de denuncia que trabajan de forma regional o estatal. Tienen por objetivo la lucha ante las injusticias y movimiento en defensa de los derechos y libertades, para dar visibilidad, crear reacciones generativas contra los hechos denunciados, que den lugar a soluciones y propuestas constructivas, y en algunos casos facilitar soporte jurídico. A la hora de denunciar, se debe utilizar todos los recursos disponibles, y las tecnologías sociales son uno de los más contundentes. Sin embargo, la arquitectura de las plataformas las hace susceptibles a manipulaciones,y además se pueden comprar valoraciones positivas.Todo ello resta credibilidad a la información y disminuye la confianza de los usuarios en la plataforma.

Entre los sistemas que incorporan la tecnología blockchain para la gestión de denuncias con el objetivo de dar solución a los problemas planteados, encontramos una plataforma contra el acoso laboral hacia las mujeres y otra aplicación de la policía en India para el registro de denuncias.

En el cuerpo de policía de la India la ley establece que las denuncias deben quedar registradas. En la práctica, tanto el aumento de actividades delictivas como la presencia de policías corruptos, que piden sobornos por el registro de los FIR(first information report), han ocasionado que cerca del 25 % de los denunciantes no pudieron registrar las denuncias, y de los que registraron el 30 % no recibió copia alguna.

Motivados por plantear solución a tal problema se propuso incorporar un sistema de gestión de denuncias policiales, E-police system[22], que utilice la tecnología blockchain para gestionar los FIR y NCR de forma descentralizada. Hacen uso de IPFS para llevar la cuenta de todas las actividades relacionadas con las denuncias policiales, desde la presentación de la denuncia hasta la entrega del

acta de acusación al tribunal. El uso de la tecnología blockchain garantiza la confianza entre los denunciantes y el departamento de policía. El sistema no sólo es seguro frente a la pérdida de datos, sino también frente a la piratería de fuerza bruta u otros tipos de ataques maliciosos.

Ethereum, gracias a la encriptación, proporciona transparencia al tiempo que garantiza la privacidad de los datos confidenciales. Los detalles de la denuncia se cifran por un algoritmos simétrico y las pruebas proporcionadas por el usuario se almacenan en una red pública IPFS. El hash correspondiente a las pruebas, los detalles de la denuncia cifrados se almacenan en la red blockchain. Una vez que la policía presenta el FIR/NCR/Hoja de cargo, se convierte en un pdf que queda posteriormente encriptado. Este pdf cifrado se almacena en la red IPFS y el hash correspondiente se almacena en la red Ethereum.

Uno de los mayores obstáculos para las mujeres en sus carreras es el acoso laboral, desde el físico, al verbal o el no verbal, que afectan negativamente a la salud de la víctima, pero también a la reputación de la empresa u organización donde sucede. Por lo general, los sistemas centralizados tradicionales no se consideran fiables ni seguros, porque no garantizan la autenticidad ni la fiabilidad de las denuncias anónimas que puedan contener información sensible de la víctima. La forma de presentar una denuncia es o bien acudir a la autoridad competente o bien rellenar un formulario de denuncia o enviar un correo electrónico. Estos métodos siembran la duda en la víctima sobre el beneficio que obtienen de su queja, porque temen los problemas que podría ocasionarles si su identidad llegara a revelarse o sus pruebas llegaran a a manipularse. Por consiguiente, los acosadores se libran con mucha facilidad.

Se busca una plataforma que garantice a las mujeres un entorno de trabajo saludable y que proporcione todo el apoyo necesario para enfrentarse a las injusticias laborales. Dado que el sistema de gestión de denuncias de acoso laboral debe ser anónimo, fiable, seguro, transparente y a prueba de manipulaciones, para brindar una solución al problema, se plantea la implementación de una plataforma basada en tecnología blockchain, que cumple todos los requisitos, donde las mujeres pueden denunciar fácilmente a sus acosadores. Introduce un modelo jerárquico de dos niveles, uno de Recursos Humanos (RRHH) que contempla las denuncias anónimas y otro para que las víctimas denuncien con su identidad revelada a la Autoridad Superior cuando se trate de algo especialmente grave[23].

3.5. Sistemas de reputación

Los sistemas de reputación buscan generar confianza a través de la reputación[24]. Para que estos funcionen con eficacia es necesario:

- Que se espere un vida útil prolongada y cree expectativas de interacciones futuras.
- Que capturen y distribuyan comentarios sobre interacciones anteriores.
- Que exista una continua retroalimentación para guiar la confianza, porque sin comentarios desaparecería el entorno de confianza.

Algunos de los problemas que plantean los comentarios son:

- La disposición del usuario a proporcionar feedback cuando no se requiere, pues en tal caso no se recopilarían suficientes comentarios.

- El usuario se vea reacio a brindar comentarios negativos por miedo a posibles represalias si no se garantizara el anonimato.
- Falta de honestidad en los comentarios de los usuarios.

Aunque no es garantizable, si se establece una comunidad de comentarios honestos es mas probable que los nuevos usuarios sean también honestos. Al fin y al cabo la mejor forma de mitigar las falsas denuncias o calumnias consiste en dar credibilidad.

La reputación se convierte en una especie de recurso preciado que ofrece una imagen de calidad. Mantener la credibilidad de estos sistemas es una tarea compleja, puesto que se ven expuestos a un gran número de ataques contra la reputación en forma de blanqueo, calumnias, DDoS , Sybil para auto-promoción, multicuentas, etc. Para defenderse se emplean mecanismos basados en IP que previenen la identidad múltiple, y otros que buscan mitigar tanto la generación como la difusión de falsos rumores. A su vez, la credibilidad de una reputación es mayor si depende de un mayor numero de usuarios.

La gran parte del estudio sobre reputaciones se centra en sistemas estables y no contemplan la posibilidad de implementarlos en sistemas descentralizados más escalables. Existen interfaces o estándares que permiten incorporar un token no transferible que sirva para representar reputación dentro del sistema[25], siguiendo el esquema general de ERC20[26].

Un sistema de reputación referente es el denominado sistema de reputación beta[27], que se basa en el uso de funciones de densidad de probabilidad beta para combinar comentarios y obtener calificaciones. Su principal ventaja es la flexibilidad y la simplicidad, así como su fundamento estadístico. Enfrenta reseñas o feedbacks positivos con los negativos para obtener una distribución de probabilidad que le permite estimar la reputación con la esperanza matemática. Combina distintas opiniones de forma muy sencilla.

FarMed [28] propone un marco inteligente que ejecutará sistemas de reputación basados en contratos inteligentes de Ethereum y desarrolla protocolos fiables basados en blockchain para trabajar con valores de reputación. El marco se divide en dos capas. La capa de datos de Blockchain donde los contratos de Ethereum verifican que las reputaciones no han sido manipuladas y se almacenan los valores de reputación. Una capa llamada AI que calcula a partir de la información de los contratos las nuevas reputaciones por medio de diferentes módulos, el cálculo de la reputación, el análisis predictivo de la reputación y el comercio de la reputación. Divide el proceso en tres fases, la fase de mercado, la fase de ejecución del contrato inteligente y la fase de cálculo del valor de confianza o reputación.

El marco se construye buscando satisfacer unos requisitos obtenidos tras un largo análisis de la literatura sobre sistemas de reputación en redes distribuidas. Estos requisitos son:

- Detección de comportamientos inadecuados de los usuarios, que soluciona por pruebas periódicas en la testnet Ropsten para así validar los métodos para detectar y tratas los fraudes de reputación.
- Modelización y deducción del valor de la reputación global basándose en los valores presentes en los contratos inteligentes , y para ello emplea un algoritmo “five-star” que calcula y actualiza las reputaciones.

- Capacidad para determinar el valor de confianza de un proveedor de servicios en un contexto. Desarrolla una ontología de servicios junto con AKTiveRank (algoritmo basado en distancias semánticas).
- Sistema de reputación como un activo digital para facilitar la creación de nuevas plataformas. Trabaja con el algoritmo EigenTrust.
- Modelos matemáticos y algoritmos que reflejen y modelen correctamente los valores de reputación, independientemente de si se siguen modelos bayesianos, de flujo, fuzzy, etc.

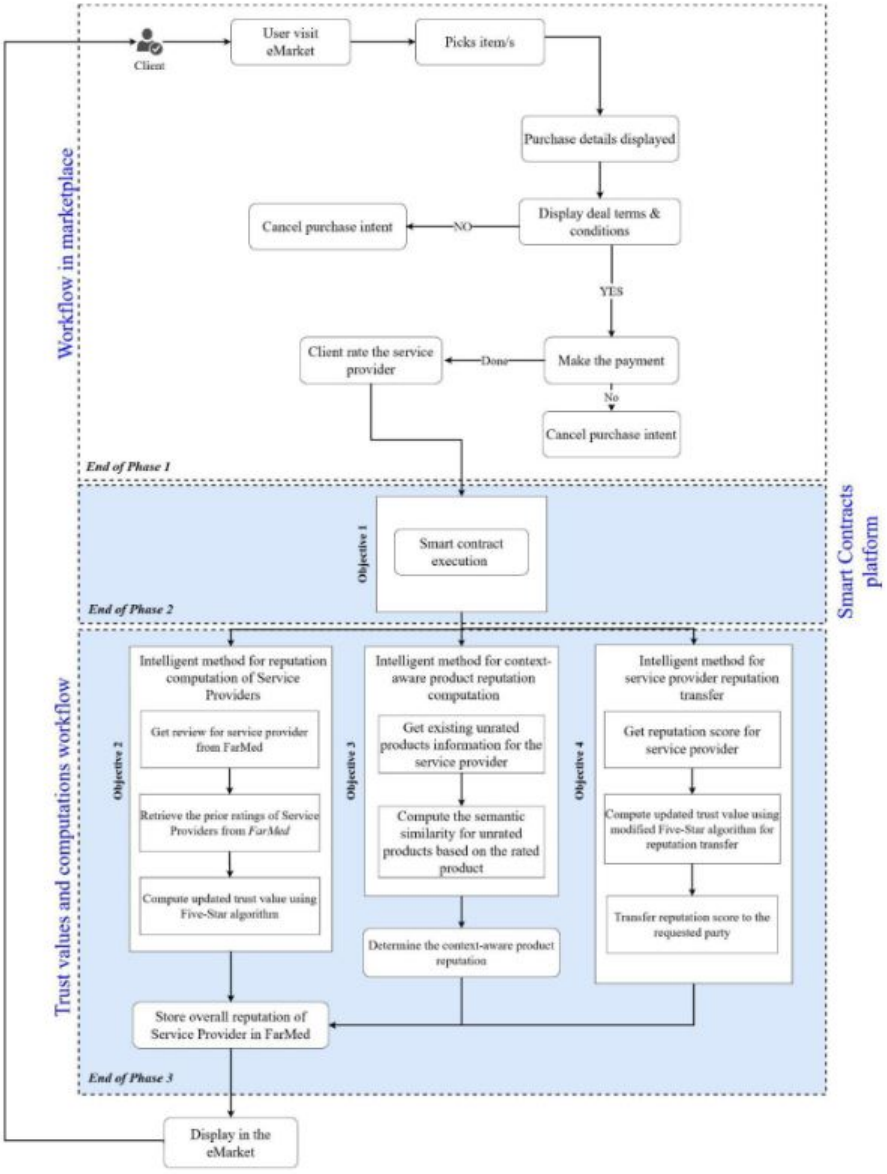


Figura 2: Marco de reputación basado en contratos inteligentes
Tomado de [28]

4. Metodología

En el presente apartado se detallará el plan establecido para la organización del desarrollo del proyecto. Se especificarán tanto las reuniones tenidas con los tutores, como la estructura interna del equipo, la repartición del trabajo y las fases llevadas a cabo hasta el desarrollo del producto final.

4.1. Planificación temporal

A la hora de planificar las tareas para el grupo, se creó un tablero de Trello donde se fueron asignando las tareas con sus respectivas fechas límite.

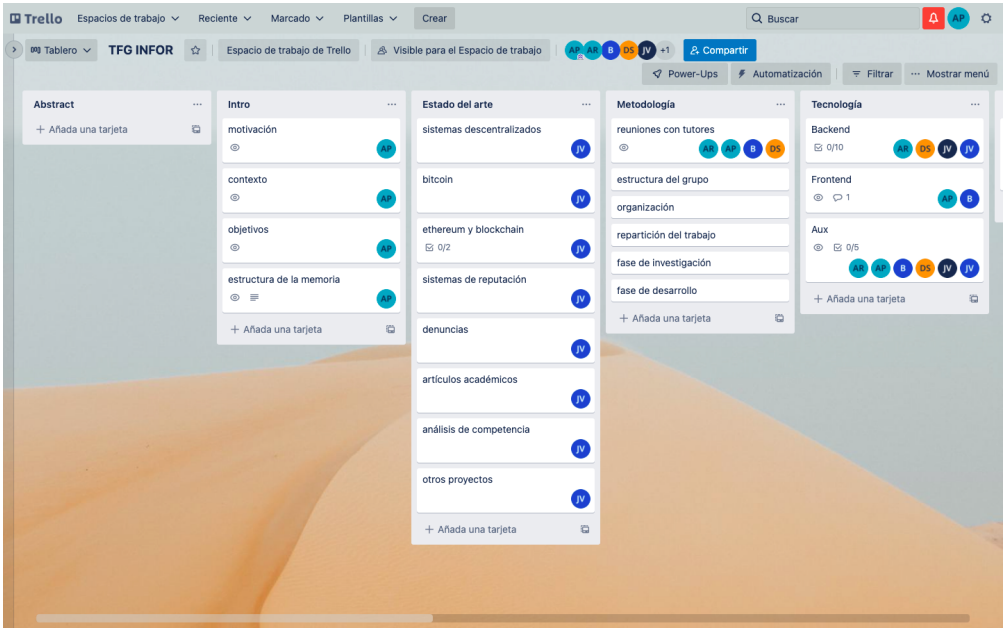


Figura 3: Planificación de las tareas en Trello
Fuente propia

Así mismo se valoró el nivel de dificultad de cada tarea y el tiempo empleado para realizarla y, en base a esto, se distribuyeron entre los miembros del equipo. La planificación temporal ha variado en función de la etapa del proyecto. En las primeras, las fechas límites eran muy flexibles, permitiendo a cada miembro del grupo llevar su propio ritmo de aprendizaje. Una vez adquiridos los conocimientos básicos para comenzar con la aplicación, los plazos fueron más estrictos. A lo largo de todo el proyecto se han tenido constantes reuniones para fijar conceptos, elaborar brainstormings y resolver problemas. En todas ellas estaban presentes todos o casi todos los integrantes del equipo, pues la disponibilidad y entrega ha sido máxima. Estas reuniones no se incluyen en la planificación temporal.

En el *Cuadro 1* se muestran los plazos que se han seguido, si bien es cierto que en algunas situaciones no se pudieron cumplir. Mientras que algunas son solo lo que indica el título, como por ejemplo desplegar el contrato en la red de prueba Rinkeby, otras son más complejas y requieren mucho esfuerzo. Este es el caso de la confección del Front-End, es decir, las interfaces que componen

la aplicación web. Todas las tareas aparecen en el Trello mostrado anteriormente. Aquellas que no se completaban en su periodo fijado, se marcaban con un color distinguido. Si además no se pudieron finalizar por errores complejos, recibían otro color.

Fases	Tareas	Plazos	Plazos totales
Fase de aprendizaje e investigación	Investigación sobre Blockchain, Bitcoin y Ethereum	1/10 - 15/10	Octubre - Noviembre
	Investigación sobre sistemas de reputación y métricas de discriminación	16/10 - 31/10	
	Investigación sobre denuncias, GDPR y tipos de discriminación	1/10 - 31/10	
	Prácticas introductorias de Solidity	1/11 - 15/11	
	Prácticas introductorias de React	16/11 - 30/11	
Primera fase de desarrollo	Primer smart contract	1/12 - 15/12	Diciembre - Enero
	Organizar y generar la base de datos para el sistema centralizado	1/12 - 23/12	
	Primeras interfaces de usuario	1/12 - 23/12	
	Lanzar smart contract a la red Rinkeby	1/01 - 15/01	
	Servidor, API LinkedIn y conectar con la base de datos	1/01 - 31/01	
	Establecer conexión BACK-FRONT	15/01 - 31/01	
	Mejorar interfaces de usuario y confeccionar nuevas	1/01 - 31/01	
Completar versión inicial del proyecto	Mejorar el smart contract	1/02 - 20/02	Febrero - Marzo
	Elaborar todas las interfaces de la aplicación	1/02 - 31/03	
	Revisar todas las funcionalidades	15/03 - 31/03	
	Comenzar memoria del proyecto	1/02 - 31/03	
Versión final del proyecto	Introducir tokens reputacionales en smart contract	1/04 - 7/04	Abril
	IPFS y conexión con smart contract	7/04 - 15/04	
	scraping para obtener reputaciones externas	1/04 - 7/04	
	Incluir Docker para portabilidad	1/04 - 15/04	
	Modificar base de datos para incluir reputaciones externas	1/04 - 15/04	
	Establecer conexión BACK-FRONT	15/04 - 23/04	
	Modificar interfaces de usuario	1/04 - 23/04	
	Añadir nuevas funcionalidades respecto a la versión inicial	1/04 - 23/04	
	Revisar funcionalidades y posibles errores	23/04 - 31/04	
	Completar memoria del proyecto	1/04 - 3/05	
Evaluación con usuarios y correcciones finales	Diseñar plan de evaluación	3/05 - 5/05	
	Seleccionar los participantes	5/05 - 6/05	
	Realizar las entrevistas	6/05 - 8/05	
	Analizar entrevistas y revisar feedback de los usuarios	9/05 - 11/05	
	Realizar correcciones de la versión final	11/05 - 14/05	

Cuadro 1: Planificación temporal resumida
Fuente propia

A continuación se detalla la primera fase, de aprendizaje e investigación. Aclarar que esta fase fue esencialmente individual, combinada con reuniones esporádicas para compartir conocimientos adquiridos. Así pues, fue decisión individual de cada miembro qué libros leer o qué cursos seguir. Agradecer a los tutores el material aportado, ya que era de gran calidad y permitió cumplir con creces el objetivo de esta fase inicial. Se profundizará sobre el resto en los capítulos relativos a la propia aplicación web y el desarrollo de las versiones.

- **Investigación sobre Blockchain, Bitcoin y Ethereum**:

  El conocimiento de esta rama era muy variopinto entre los miembros del equipo. Algunos desconocían por completo la materia, mientras que otros habían cursado la asignatura *Introducción a la Tecnología Blockchain y Smart Contracts*, ofertada como asignatura optativa por la Facultad de Informática de la UCM. Todos los miembros leyeron los artículos ofrecidos por los tutores. Para reforzar con otros medios, algunos también hicieron un par de cursos de la plataforma Udemy: *Blockchain de cero a experto* y *Ethereum and Solidity: The Complete Developer's Guide*. Otro curso que también fue de gran utilidad, recomendación directa de nuestro co-tutor de Telefónica Carlos Pastrana, para aprender el lenguaje de programación Solidity es el de la página web *Cryptozombies*.

  Blockchain y Bitcoin sirvieron como trampolín para poder comprender Ethereum, pues es la plataforma sobre la que gira todo el proyecto. Una vez entendidas ambas tecnologías, el siguiente paso era comprender los principales y novedosos conceptos de Ethereum: **Smart Contract y DAOS**. Estos conceptos, nos fueron introducidos por parte de nuestro tutor Samer y, posteriormente, fuimos investigando para conocerlo en profundidad, ayudados siempre de los cursos que realizamos. Además, todos los miembros tuvieron una buena formación en el lenguaje de programación **Solidity**, lenguaje usado para la implementción de los Smart Contracts en la red de Ethereum.

- **Investigación sobre sistemas de reputación y métricas de discriminación**:

  Para decidir cómo sería el diseño del sistema de reputación, los miembros del equipo realizaron una investigación sobre qué aspectos se debían tener en cuenta a la hora de desarrollar uno y, sobre algunos ejemplos tanto en la Blockchain como en otras plataformas centralizadas. Tras todo el recabado de información hecho por el equipo durante un par de semanas se llegó a una conclusión conjunta: el dilema sistémico al que se enfrentaban aquí era encontrar un difícil equilibrio entre funcionalidad y fiabilidad.El sistema debe ser lo suficientemente robusto para evitar manipulaciones pero, además, debe ser transparente y confiable para que la gente lo use. Estos dos objetivos, parecen algo contradictorios en primera instancia ya que al intentar construir un sistema infalible, que haga imposible cualquier manipulación, seguramente éste sea inoperativo por su gran complejidad. Las decisiones claves que se tomaron para el diseño del mismo fueron las siguientes:

  - Acciones: qué acciones del comportamiento de las empresas son relevantes para la cuantificación de la reputación. Es decir, qué acciones, hechos o información es necesario registrar y monitorizar.
  - Fuentes de información: Una vez identificadas las "acciones" que se quieren monitorizar, es necesario decidir cómo obtener la información de cada una. A menudo hay que optar entre un mecanismo "implícito" o "explícito", o sea, entre datos generados internamente por el sistema o por el feedback aportado explícitamente por los usuarios.En nuestro

caso, se decidió por un híbrido entre ambos ya que se tienen en cuenta datos generados por el sistema como son las denuncias interpuestas por usuarios y datos procedentes de valoraciones externas hechas por usuarios en páginas como Indeed obtenidas mediante scraping.

- Modelos de agregación: una de las decisiones más importante a tomar en un sistema de reputación es cómo agregar y representar la información reputacional para conseguir el efecto esperado en los usuarios de la plataforma. Tras varias reuniones y lluvias de ideas se decidió que la mejor forma de representar la información reputacional recogida era usar ***dimensiones múltiples*** para juzgar a las empresas de modo que los usuarios puedan hacerse una opinión de una empresa particular tras valorar distintos atributos. Para representar los valores de reputación se han usado diferentes mecanismos: por un lado se representa la reputación de forma numérica en una escala de 0 a 10 y en otros casos, se representa gráficamente con estrellas (de 0 a 5). Finalmente, en relación a los modelos de agregación se decidió revelar la información de la fórmula de agregación de la reputación puesto que se pretende generar confianza entre los usuarios de la plataforma.
- Mecanismos de protección: un punto crítico es decidir cómo se va a gestionar la identidad de los usuarios. Apostar por el anonimato puede facilitar las prácticas negativas, mientras que controlar la identidad suele disuadir a que muchos usuarios participen o que los que participen lo hagan con un sesgo demasiado positivo. Por ello y como solución óptima se planteó que la identidad del usuario fuese conocida por el sistema pero no revelada a la comunidad. Se propone un **anonimato autenticado**. Por ello, para poder denunciar a una empresa es necesario entrar en la cuenta de LinkedIn para verificar una identidad pero ninguno de los datos personales del usuario es mostrado ni recogido por la plataforma. En próximas secciones se explicará con más detalle el proceso seguido para implementar este anonimato autenticado.

Finalmente, es necesario añadir que tras varias reuniones realizadas con Silvia se decidieron recoger de forma anónima datos cuantitativos y cualitativos de las denuncias interpuestas a nivel general y a nivel empresa con el fin de presentar gráficas sobre número de denuncias según rango de edad, según género, etc.

- **Investigación sobre denuncias, GDPR y tipos de discriminación**:

  Para desarrollar las bases fundamentales de nuestro proyecto, se hizo una investigación sobre los tipos de discriminación que se podían producir en el lugar de trabajo, con el objetivo de recoger todos los posibles. En este punto se invirtieron bastantes esfuerzos buscando y recabando información de proyectos, plataformas, y páginas webs similares, para tratar de entender los puntos principales que debíamos tener en cuenta y las necesidades básicas que era necesario cubrir en una plataforma que pretende ayudar y servir de eco a personas en una situación desfavorable tras sufrir discriminación de algún tipo. Además, se visitaron las páginas web de las comisiones de derechos humanos de distintos países que cuentan con servicios de denuncias para estudiar cómo abordaban en estos casos la recogida de datos de los denunciantes.

  Con respecto a este último punto, los miembros del equipo tenían ciertas nociones sobre el GDPR tras haber cursado la asignatura de ELP y para asegurar que se cumplía la normativa se revisaron ciertos puntos del reglamento. Además, siempre teniendo en cuenta el espíritu de una plataforma desarrollada en la Blockchain, que se basa en el anonimato de los usuarios y la transparencia de la misma. Todo este proceso se realizó siempre contando con la opinión de nuestra tutora Silvia para perfilar y perfeccionar todos los detalles relacionados con qué

datos era oportuno y cuáles no recoger en la denuncia, cuáles debían pedirse obligatoriamente y cuáles se debían dejar a libre elección del denunciante y qué tratamiento se les debía dar a los mismos. En estos aspectos, contar con la opinión de una socióloga fue fundamental para el óptimo diseño del formulario de denuncia y la posterior presentación de gráficos y métricas con la información recabada de las denuncias.

- **Prácticas introductorias de Solidity**:

  La mayoría de los integrantes del equipo no conocían Solidity por lo que tutor proporcionó una práctica introductoria a este lenguaje consistente en el desarrollo de un sorteo por medio de un **smart contract**. El funcionamiento deseado del contrato era el siguiente: varias personas podían entrar a participar en un sorteo tras pagar una cantidad de dinero previamente establecida al contrato y, al resolverse el sorteo, todo el dinero recolectado se enviaría a uno de los participantes elegido al azar. Así, se aprendió a desarrollar en un lenguaje de programación nuevo **(Solidity)**, gestionando envío y recepción de criptomonedas **(ether)**, interactuando con un software online sin servidores **(desplegado en una blockchain)**, y probándolo desde una cartera propia de criptomonedas integrada en el navegador **(Metamask)**.

- **Prácticas introductorias de React**:

  Al igual que en el punto anterior, no se había trabajado previamente con React por lo que los miembros del equipo realizaron una pequeña práctica consistente en el diseño de un *To do List*. En este caso, se dejó un poco a elección propia el diseño de la lista para *cacharrear* en investigar con diferentes componentes, funcionalidades y estilos en React. Los resultados de estas prácticas fueron muy diferentes de unos miembros a otros y sirvió para realizar un acercamiento a cómo diseñar una interfaz gráfica propia al gusto del programador.

4.2. Software Libre

Para llevar a cabo el desarrollo del proyecto se decidió alojar el código en un repositorio público de GitHub, de forma que todos los integrantes pudieran colaborar y modificar los ficheros además de aprovechar las herramientas de control de versiones que GitHub pone a dispoción de los usuarios. El **repositorio** creado y su contenido pueden verse en: https://github.com/javVM/TFGI-Plataforma-blockchain-para-visualizar-la-reputacion-de-empresas-sobre-casos-de-discriminacion.

Con el objetivo de que nuestra aplicación pudiera ser de software libre, nos aseguramos de que las distintas librerías y software utilizado hubieran sido licenciadas bajo licencias de software libre. Las licencias que emplea el distinto software de terceros son las siguientes:

- Licencia MIT.
- Licencia Apache versión 2.0.
- Licencia BSD 2-Clause.
- Licencia BSD 3-Clause.

Para poder cumplir con los requisitos de estas licencias se creó un fichero llamado *ThirdParty-Notices.txt*, en el que se declara todo el software de terceros utilizado y se copian los respectivos textos de licencia y atribución de autoría siguiendo los requisitos de las licencias listadas.

Finalmente, tras estudiar los distintos tipos de licencia que se podrían utilizar para licenciar nuestro proyecto se optó por emplear *GPL v3*. Esta licencia es compatible con las licencias mencionadas y permite a todos los desarrolladores poder emplear nuestro código en sus proyectos libremente siempre y cuando este no sea propietario. Consideramos que esta licencia cumple con la meta de poder contribuir a la creación de una comunidad que brinde apoyo a los denunciantes mediante el desarrollo de plataformas basadas en lo implementado en este proyecto. En adición, al ser público el repositorio, se da cabida a que cualquier persona interesada pueda acceder, realizar un fork para introducir mejoras y/o incorporarlo a su proyecto.

4.3. Organización

A la hora de organizar las tareas y cómo se iba a desarrollar el trabajo, lo primero que hicimos fue crear un Miró compuesto por post-it que nos permitiese hacer brainstorming sobre qué aspectos sería interesante incluir en la plataforma. La metodología fue la siguiente: cada elemento principal tiene un post-it morado y a su alrededor se sitúan post-it amarillos con las ideas para ese elemento. Más adelante, a medida que se desarrolla el proyecto, estos últimos cambian de color según haya sido posible su realización (verde), no haya sido posible (rojo) o se deje para una mejora futura (amarillo). En cuanto a los elementos principales, Back-End y Front-End tienen colores morado oscuro y claro respectivamente. Todo esto queda reflejado en la imagen:

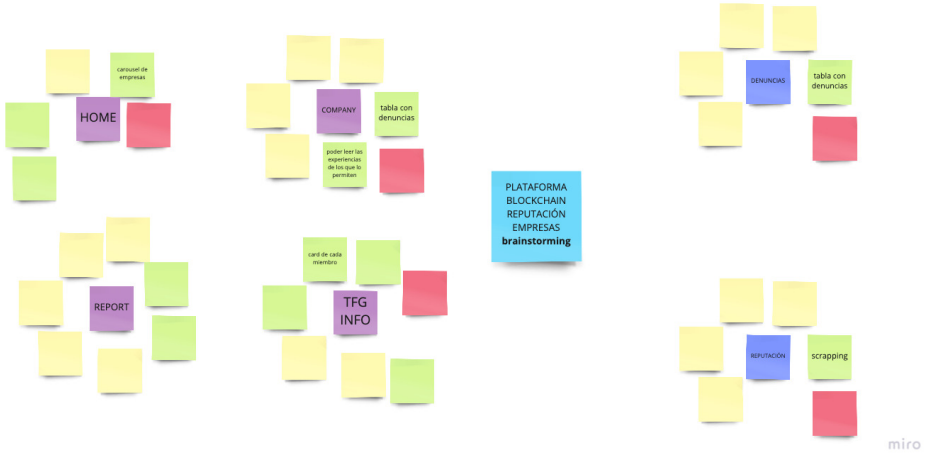


Figura 4: Brainstorming en Miró
Fuente propia

Una vez hubimos plasmado sobre Miró las ideas principales, procedimos a darles forma y buscar información sobre cómo podríamos implementarlas, tratando de bajar a tierra aquellas que eran posibles y descartando el resto. Todo esto lo fuimos apuntando en un documento de Google Drive, usando como título para cada sección el nombre que aparecía en los cuadros morados del miró. De esta manera es más fácil ver cómo se han desarrollado las ideas.

Hecho esto, llegó el momento de sacar tareas específicas que a posteriori pudiésemos marcar como completadas. Para cada uno de los elementos principales creamos una checklist, basándonos tanto en el Miró como en el documento de Drive. Ordenamos cronológicamente estas listas para aquellos casos en que fuese necesario.

Tras esta primera fase de organización, procedimos a crear un tablero en Trello con una lista para cada una de las ideas principales, incluyendo en dicha lista las tareas de la checklist, una por tarjeta. Asimismo, cada tarea podía quedar dividida en pequeñas checklist si era necesario. Por último, asignamos las tareas a cada miembro con la misma aplicación, como se ha explicado en la sección de planificación temporal.

De esta forma, podíamos tener un control total y exhaustivo de qué tareas habían sido completadas y cuáles quedaban por realizar.

4.4. Repartición del trabajo

Llegado el momento de repartir las tareas que obtuvimos durante la primera fase de la organización, que como expusimos anteriormente, quedaron reflejadas en el tablero de Trello, procedimos a asignarlas basándonos en las preferencias de cada uno y en sus conocimientos. Decidimos repartirnos en 2 grupos diferentes, uno encargado de la parte de Back-End, el más numeroso, constituido por Alejandro Ramírez, David Seijas, Javier Verde y Jorge del Valle, y otro del Front-End, formado por Ana Belén Duarte y Ángeles Plaza.

Dentro de cada uno de estos subgrupos, la repartición del trabajo se realizó de forma interna.

4.4.1. Front

En el caso del Front, nos pareció más adecuado y fácil de gestionar realizar una repartición de las pantallas de la plataforma, exceptuando el formulario. De esta manera, sería más fácil la cohesión de las versiones de la aplicación, pues cada persona trabajaría de forma unívoca sobre cada pantalla, lo que facilitó el trabajo. Asimismo, para realizar muchas de las funcionalidades, como los dos tipos de gráficas, se repartió el trabajo equitativamente.

Se realizaban pequeñas reuniones de forma continuada y casi diaria, en la que se compartían los nuevos cambios para tratar que la cohesión del diseño entre las diferentes pantallas fuese óptima.

Posteriormente, tuvimos que realizar una nueva versión de la plataforma, y por tanto repetir por completo el diseño, basándonos en una plantilla. La repartición en este caso fue similar, tratando de reciclar al máximo los componentes de la aplicación anterior, y asignando prácticamente las mismas tareas.

4.4.2. Back

Como ya se ha indicado anteriormente, el Back-End está claramente dividido en una parte centralizada y otra descentralizada. Debido a sus conocimientos sobre servidores, bases de datos y APIs, Javier Verde se encargó de la parte centralizada. Además, elaboró el scraping sobre Glassdoor e Indeed para obtener las reputaciones externas a nuestra aplicación, que ofrecen una mayor robustez inicial, así como los Dockerfiles y archivo Docker Compose para hacer posible la dockerización de la aplicación y facilitar su portabilidad.

El resto de los integrantes destinados a esta rama se dedicaron al sistema descentralizado. Las principales tareas eran: confeccionar un smart contract que incluyera tokens para controlar la reputación de las empresas; almacenar la información de cada denuncia en otra red descentralizada, en nuestro caso IPFS; y conectar todo el sistema descentralizado con Front-End, de forma que se

extrayeran los datos de las denuncias del formulario de la plataforma y almacenarlos de manera segura en la Blockchain y, de manera inversa, extraer luego estos datos de la red de Ethereum e IPFS para poder utilizarlos en nuestra plataforma para la muestra de reputaciones y gráficas.

En cuanto a la parte descentralizada, los 3 miembros que trabajaban en esta se dividían las tareas que tocaban en cada momento e intentaban avanzar por su cuenta, aunque la mayoría de días trabajaban de manera conjunta y síncrona mientras hacían una videollamada por Discord para coordinarse y solucionar los problemas que surgía de manera conjunta. De esta manera, cada uno trabajaba e distintas tareas, pero complementándose, o investigando conjuntamente para hallar la mejor forma de realizar alguno de los cometidos.

Aproximadamente cada semana los 4 miembros se reunían vía Discord para mostrar los avances. Para preservar la consistencia de ambos sistemas, en caso de dudas siempre se trataban de resolver lo más rápido posible. Desde nuestro punto de vista, la comunicación y la rápida solución de problemas/dudas es la clave para obtener buenos resultados.

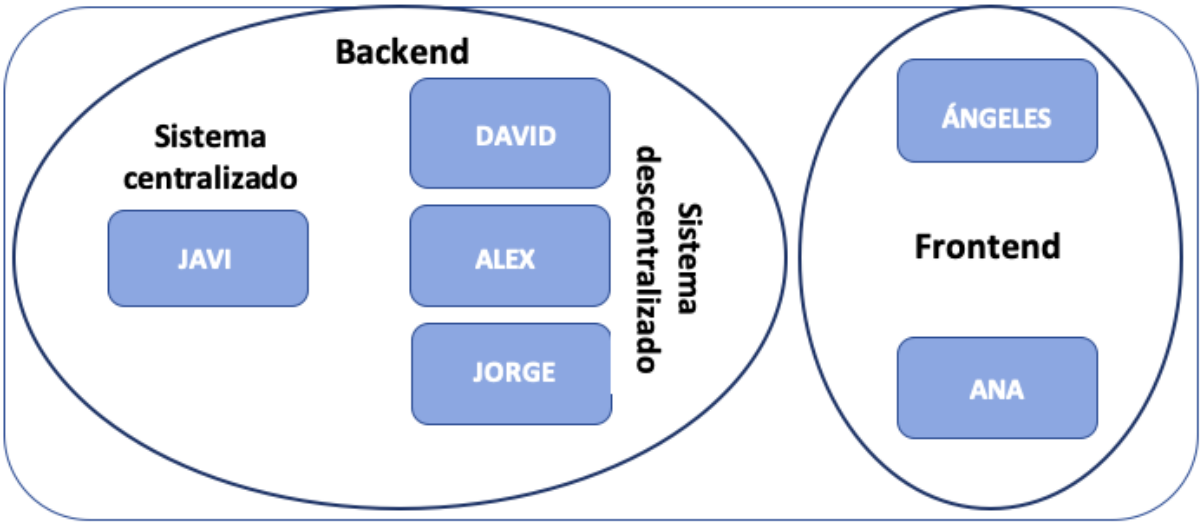


Figura 5: Estructura del equipo
Fuente propia

4.5. Reuniones con los tutores

Para el seguimiento del desarrollo del proyecto, así como para solventar las dudas que el equipo fue teniendo a lo largo del mismo, se realizaron en primera instancia, reuniones cada dos semanas con los tutores del TFG y con el co-tutor. Esta dinámica se siguió hasta las vacaciones de Navidad y enero donde se realizó un parón debido a los exámenes de la primera convocatoria. Al finalizar el mes de enero, se retomaron las reuniones para fijar nuevos objetivos y continuar el proyecto, aunque de forma menos asidua, pues el equipo tenía a la vista los siguientes pasos a seguir. Estas reuniones se realizaron telemáticamente usando Google Meet y Teams.

4.6. Estructura del equipo

Antes de establecer el tipo de estructura de equipo utilizado, presentamos los siete factores que determinan un tipo u otro establecidos por Mantei. En el caso de nuestro proyecto resultaría como en el cuadro siguiente:

Tamaño del equipo	Pequeño, está formado por 6 personas
Dificultad del proyecto	Dificultad alta, es un proyecto a gran escala a pesar de estar todo muy estructurado
Duración del equipo	El equipo será el mismo desde el principio hasta el final del proyecto (obviando abandonos imprevistos)
Modularidad del proyecto	El proyecto presenta bastante modularidad y diferenciación en backend y frontend, aunque no son independientes. Es necesario implementar la conexión entre ellos.
Fiabilidad	El proyecto ha de ser muy fiable y funcionar correctamente desde su puesta en marcha. Esto es muy importante debido a los fundamentos de la Blockchain.
Fecha de entrega	La fecha de entrega es estricta. Final de curso (mayo de 2022)
Comunicación	La comunicación entre los miembros del grupo es muy alta, todos aportan ideas y toman decisiones en consenso.

Figura 6: Factores de Mantei
Fuente propia

Atendiendo a las características mostradas en la tabla anterior y centrándonos sobre todo en la importancia de la comunicación y el intercambio de ideas entre los miembros del grupo y la dificultad del problema, nos decantamos por una estructura del equipo Descentralizada Democrática. Sin embargo, también consideramos importante la realización de subgrupos que resuelvan problemas concretos.

5. Tecnologías

Una aplicación web requiere múltiples tecnologías para un resultado satisfactorio. No solo a nivel de Front-End, las cuáles suelen estar muy desarrolladas y existen numerosas opciones, si no también a nivel de Back-End. En este caso, al tratarse de una plataforma descentralizada, basada en blockchain, las tecnologías son escasas y se encuentran en diferentes estados de desarrollo. Además, se han utilizado otras aplicaciones para favorecer la estructura y organización del equipo y del código. A continuación se exponen todas las empleadas, separadas en los 3 estratos mencionados: Back-End, Front-End y Auxiliares. Para cada tecnología se especifica el uso que ha tenido en el proyecto.

5.1. Tecnologías Back-End

Como ya se ha especificado anteriormente, en el Back-End existen dos partes diferenciadas, cada una con su propia funcionalidad: sistema descentralizado y centralizado. El sistema descentralizado es el gestor de la reputación de las empresas, encargado de almacenar los tokens y las denuncias emitidas. El sistema centralizado permite a los usuarios conectarse a la aplicación a través de LinkedIn, así como almacenar y proporcionar las reputaciones externas de otras páginas. Además, la base de datos también almacena datos generales de las denuncias para mostrar las gráficas principales.

Por lo tanto, al ser dos campos tan diferentes, las tecnologías también lo son. Primero se exponen las asociadas al sistema descentralizado y posteriormente las relacionadas con el sistema centralizado.

5.1.1. Tecnologías descentralizadas

La base del proyecto es el sistema descentralizado. La combinación de Ethereum e IPFS es una gran solución a ello, pues ambas son tecnologías open-source, contrastadas dentro del universo Blockchain.

Solidity

Figura 7: Logo de Solidity
Tomado de [29]

Lenguaje de programación de alto nivel que permite confeccionar smart contracts sobre la red de Ethereum. Ethereum incluye una máquina virtual que ejecuta código de bytes. Se denomina **EVM**. Solidity se encarga de compilar los smart contracts en dicho formato, conocido como EVM bytecode.

Este lenguaje es Turing completo, es decir, que tiene el mismo poder de computación que una máquina de Turing. Esto significa que puede realizar las mismas operaciones que cualquier lenguaje de programación a los que estamos acostumbrados a utilizar.

Para solventar el problema de la computación infinita (programas que no terminan y consumen muchos recursos para nada), Ethereum incluye la noción de gas. Así pues, lanzar un contrato sobre la Mainnet de Ethereum o realizar cualquier acción sobre ella requiere un coste económico. Por lo tanto, este proyecto está íntegramente localizado en una red de pruebas, **Rinkeby**. En esta, los ethers que se consumen son ficticios.

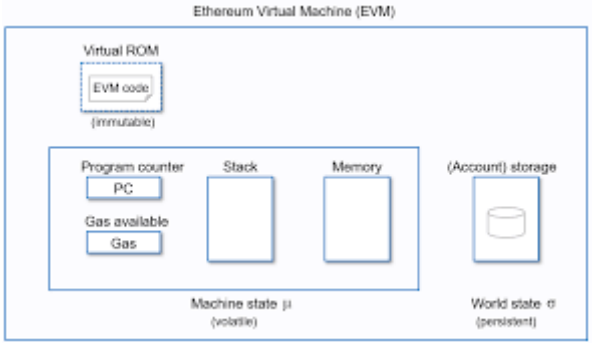


Figura 8: Ethereum Virtual Machine (EVM)
Tomado de [30]

Rinkeby

Figura 9: Logo de Ethereum
Tomado de [31]

Red de prueba de Ethereum en la que no se gastan ethers reales. Los ethers ficticios se obtienen a través de Rinkeby Faucet [32]. Todos los contratos han sido lanzados a esta red. De esta forma, las denuncias realizadas por nosotros para probar el funcionamiento de nuestra plataforma son transacciones en esta red que no se reflejarán en la red real de Ethereum.

ERC

Estándares especificados por la propia comunidad de Ethereum para crear tokens con propiedades comunes y que sean interoperables entre diferentes contratos. Los más conocidos son ERC-20, el estándar de los tokens fungibles; y ERC-721, el de los tokens no fungibles, más conocidos por su abreviatura NFT's.

El sistema de reputaciones de la aplicación está basado en tokens. Como no son únicos para cada empresa, el formato que más se ajusta es el de los tokens fungibles. Sin embargo, el principal problema es que son transferibles, lo que choca frontalmente con el concepto de reputación de una empresa.

Dentro de los estándares ya admitidos por toda la comunidad, no existe ninguno que implemente tokens no transferibles. Es por ello que hemos decidido utilizar un estándar que esta en aras de ser aprobado.

EIP

Propuestas de estándar de la comunidad de Ethereum. Nuestros tokens están implementan una de estas interfaces: EIP - 4974. Escogimos este estándar porque se ajusta perfectamente a los requisitos: fungibles y no transferibles. Existen otras propuestas como EIP - 4671, aunque desde la propia página de Ethereum desaconsejan su uso porque están modificando la implementación.

OpenZeppelin

Figura 10: Logo de OpenZeppelin
Tomado de [33]

Librería de código abierto para Solidity. Proporciona múltiples implementaciones, como por ejemplo ERC-20. Además, incluye otro estándar, ERC-165, que permite averiguar si un contrato ha implementado o no una interfaz.

Remix IDE

Figura 11: Logo de Remix IDE
Tomado de [34]

Entorno de desarrollo de código abierto que permite implementar contratos inteligentes en Solidity y desplegarlos en cualquiera de las redes de Ethereum. Además, una vez desplegado el contrato, se puede interactuar con él mediante su GUI. Consiste en ejecutar llamadas a las funciones del contrato. Estas se comportan como auténticas transacciones de la Mainnet, a pesar de estar en una red de pruebas.

Remix se ha utilizado en el proyecto para crear los contratos y comprobar todas las funcionalidades. Sin embargo, los contratos que se han utilizado en ambas versiones de la aplicación han sido desplegados desde local.

Infura

Figura 12: Logo de Infura
Tomado de [35]

Plataforma gratuita que permite a una DApp acceder a las redes de Ethereum sin la necesidad de ejecutar un nodo de forma local. Proporciona una API que ofrece acceso a un cluster de nodos públicos. Por lo tanto, el usuario no debe preocuparse del buen funcionamiento del nodo, ya se

encarga Infura. Además de la parte de Ethereum, ofrece una API para IPFS, que es la empleada en el proyecto.

Para poder utilizar la API de Ethereum, es necesario crear una cuenta en su plataforma e iniciar un proyecto. Una vez hechos los dos pasos, la página web de Infura posee una GUI que permite a los usuarios analizar las interacciones del proyecto con la red Rinkeby en nuestro caso.

Web3j

Figura 13: Logo de Web3j
Tomado de [36]

Librería de Android y Java que proporciona funcionalidades para trabajar sobre Smart Contracts y conectarse con nodos de cualquier red de Ethereum, como por ejemplo los de Infura.

Ethers.js

Figura 14: Logo de Ethers.js
Tomado de [37]

Librería con múltiples funcionalidades sobre contratos inteligentes de Ethereum. Algunas funcionalidades de la librería web3j están quedando obsoletas, por lo que conviene utilizar esta.

MetaMask

Figura 15: Logo de MetaMask
Tomado de [38]

Extensión del navegador que permite interactuar con cualquier red de Ethereum. También sirve como administrador de cuentas. Se utiliza para gestionar todos los pagos de las interacciones con el contrato.

IPFS

Figura 16: Logo de IPFS
Tomado de [39]

IPFS, siglas de *InterPlanetary File System*, es un sistema de archivo descentralizado cuyo objetivo es garantizar la seguridad y privacidad de tus datos. Impulsa una web descentralizada en la que poder almacenar archivos de todo tipo de forma segura y permanente a través de una Blockchain. De esta forma, cualquier persona puede acceder a la enorme cantidad de datos que almacena en cualquier momento. Nosotros usamos este sistema de archivos para almacenar las denuncias, guardando solo el hash de esta en el SmartContract. Así conseguimos que el almacenamiento de una denuncia sea mucho más eficiente y barato.

IPFS API

IPFS proporciona una API que permite almacenar y descargar ficheros. Infura proporciona endpoints para acceder a nodos del sistema IPFS.

5.1.2. Tecnologías centralizadas

La aplicación debe ser capaz de almacenar y servir ciertos datos, así como gestionar procesos de forma rápida por lo que es necesario el uso de herramientas que faciliten la creación de un sistema centralizado que satisfaga los requerimientos.

Mongo Atlas

Figura 17: Logo de Mongo Altas
Tomado de [40]

Base de datos [41] no relacional multicloud que permite almacenar la información dentro de clústeres. Ofrece la posibilidad de escoger distintos proveedores de servicios en la nube como AWS o Google Cloud, así como la región en la que situar las réplicas de la base de datos. A su vez, posee drivers que facilitan el acceso a los datos desde diversos lenguajes y entornos.

Node.js

Figura 18: Logo de Node.js
Tomado de [42]

Entorno de código abierto basado en JavaScript [43] que permite instalar paquetes o módulos de código listos para su uso durante el desarrollo de la aplicación. Se han utilizado diversos paquetes de funcionalidad variada durante el desarrollo del proyecto. Algunos ejemplo son:

- **node-cron:** Define un planificador de tareas que permite planificar la ejecución de un código para que se ejecute en tras cada intervalo de tiempo definido [44].

- **python-shell:** Permite la ejecución de scripts de Python a modo de proceso hijo del programa [45].

Nginx

Figura 19: Logo de Nginx
Tomado de [46]

Herramienta de software libre que permite la creación de un servidor que actúa como balanceador de carga, servidor de aplicaciones o proxy inverso, entre otras funcionalidades.

Express.js

Figura 20: Logo de Express.js
Tomado de [47]

Infraestructura de aplicaciones web [47] diseñada para Node.js que proporciona las herramientas necesarias para creación de un servidor a modo de API.

Mongoose

Figura 21: Logo de Mongoose
Tomado de [48]

Librería usada para actuar como un puente para la interacción entre servidor y una base de datos MongoDB. En adición, facilita el modelado de los datos a insertar y/o recibir de la base de datos de forma que sea posible trabajar con información estructurada.

Passport.js

Figura 22: Logo de Passport.js
Tomado de [49]

Middleware que se ejecuta cuando el usuario realiza una petición al servidor y se emplea para realizar la autenticación y autorización de un usuario mediante el inicio de sesión en distintas pla-

taformas y redes sociales como Facebook, Twitter, Microsoft o GitHub. En el caso de este proyecto, se ha utilizado a la hora de crear la identificación mediante LinkedIn.

LinkedIn API

Figura 23: Logo de LinkedIn
Tomado de [50]

LinkedIn pone a disposición de desarrolladores una API con la que obtener información básica del perfil de un usuario o empresa, ofertas de trabajo, publicaciones o campañas de anuncios.

Swagger UI

Figura 24: Logo de Swagger UI
Tomado de [51]

Herramienta de software libre cuya función es la de ofrecer documentación sobre una API. Hace posible la interacción con los endpoints que componen la API, pudiendo realizar peticiones de prueba.

Jupyter Notebook

Figura 25: Logo de Jupyter
Tomado de [52]

Aplicación web de software libre con la que se puede ejecutar código desde el navegador además de tener la capacidad de ser exportado como a modo de script.

Axios

Figura 26: Logo de Axios
Tomado de [53]

Cliente HTTP [54] que puede emplearse en Node.js o en la propia aplicación para realizar peticiones asíncronas a una API. Esta herramienta ha hecho la comunicación entre el servidor y la aplicación posible.

5.2. Tecnologías Front-End

React.js

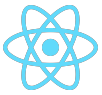


Figura 27: Logo de React
Tomado de [55]

Librería de Javascript de código abierto diseñada para crear interfaces de usuario mediante el uso de componentes que mantienen su propio estado.

Google Sheets

Figura 28: Logo de Google Sheets
Tomado de [56]

Editor de hojas de cálculo gratuito de Google.

Material UI

Figura 29: Logo de Google Material UI
Tomado de [57]

Librería de React que posee componentes [58] listos para su integración en aplicaciones web. En este proyecto también se ha utilizado Berry [59], una plantilla construida mediante esta librería

Semantic UI

Figura 30: Logo de Semantic UI
Tomado de [60]

Framework de desarrollo Front-End que permite la facilita la creación de temas además de contener componentes editables para incorporar en la web.

5.3. Tecnologías Auxiliares

En adición a las tecnologías utilizadas para el desarrollo del Front y Back de la aplicación, han sido necesarias el uso de herramientas de diversa índole que sirvieran como apoyo para la

realización de tareas como planificación del trabajo, almacenamiento del programa o comunicación con el equipo.

GitHub

Figura 31: Logo de GitHub
Tomado de [61]

Plataforma de desarrollo colaborativo que cuya finalidad es la creación y alojamiento de repositorios. Facilita el control de versiones, descarga de código y trabajo en equipo de los desarrolladores sobre un mismo proyecto.

Visual Studio Code

Figura 32: Logo de Visual Studio Code
Tomado de [62]

Editor de código de software libre desarrollado por Microsoft cuyo principal atractivo son la multitud de extensiones que tiene a su disposición para programar en distintos lenguajes y adaptarse a frameworks.

Docker

Figura 33: Logo de Docker
Tomado de [63]

Plataforma de código abierto que se utiliza para crear contenedores [64], los cuales son una instancia ejecutable de la imagen de una aplicación con todas las dependencias instaladas.

Los contenedores, a diferencia de las Máquinas Virtuales no requieren que se albergue el sistema operativo completo de la aplicación, sino que utiliza el kernel del dispositivo en el que se está ejecutando.

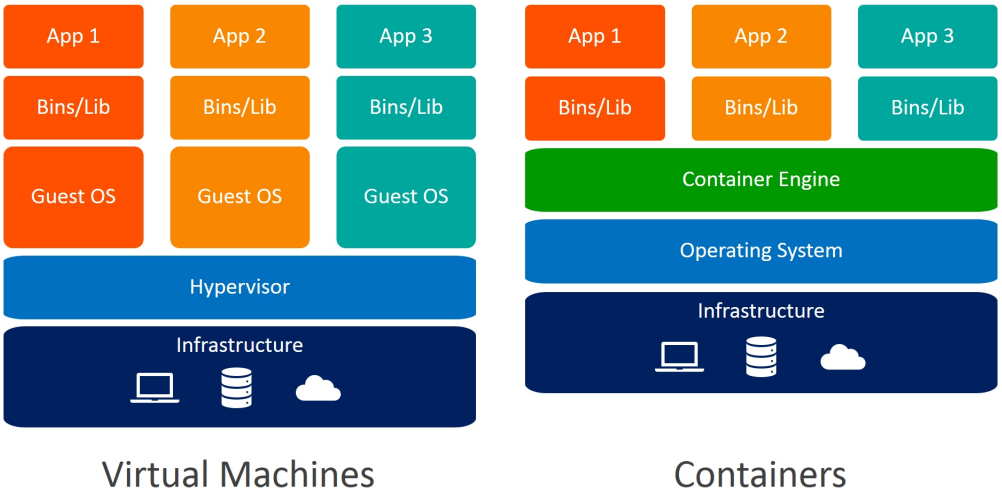


Figura 34: Contenedores vs Máquinas Virtuales
Tomado de [65]

Añade portabilidad [64] al proyecto, ya que al no poseer un sistema operativo por cada aplicación, los contenedores son muy ligeros y están listos para ser ejecutados en cualquier entorno. Si se combina con Docker Compose [66], se pueden ejecutar múltiples contenedores como un solo servicio, pudiendo desplegar una aplicación completa mediante un solo comando.

Microsoft Teams

Figura 35: Logo de Microsoft Teams
Tomado de [67]

Plataforma de comunicación y colaboración que permite chatear, crear grupos, realizar videoconferencias y la distribución de archivos editables por los distintos integrantes del equipo. Hemos usado esta plataforma para las reuniones con nuestro co-tutor de Telefónica.

Google Meet

Figura 36: Logo de Google Meet
Tomado de [68]

Servicio para videoconferencias desarrollado por Google. Esta plataforma se ha utilizado para realizar las distintas reuniones que hemos tenido con los tutores del TFG durante todo el curso académico.

Google Drive

Figura 37: Logo de Google Drive
Tomado de [69]

Servicio de almacenamiento de archivos desarrollado por Google. Hemos empleado este para realizar documentos, como órdenes de las reuniones, que necesitábamos modificar cada uno de los miembros del equipo.

Balsamiq Wireframes

Herramienta de maquetado que Google Drive tiene como extensión. Gracias a esta herramienta ha sido posible la creación de los bocetos y prototipos de cada una de las ventanas de nuestra plataforma.

Overleaf Editor de Latex online. Hemos manejado este editor para la escritura conjunta de la memoria del trabajo.

Trello

Figura 38: Logo de Trello
Tomado de [70]

Es un software de administración de proyectos usado para gestionar proyectos de una forma visual. Se pueden crear tableros con distintos trabajos a realizar recogidos en tarjetas. Las tarjetas se pueden asignar a los miembros de proyecto de forma que haya una división de la carga de trabajo, además de dejar constancia de esta repartición. Es posible añadir checklists dentro las tarjetas para indicar si una tarea ha sido realizadas, adjuntar documentación y establecer una fecha límite para acabar la parte asignada.

Miró

Figura 39: Logo de Miró
Tomado de [71]

Pizarra colaborativa cuyas funcionalidades permiten llevar a cabo tareas como documentación, realización de diagramas o brainstorming; siendo esta última el motivo por el que se le ha dado uso en este proyecto. Contiene varias herramientas como notas adhesivas sobre las que hacer anotaciones, formas para diagramas de flujo y chats de texto y vídeo.

Figma

Figura 40: Logo de Figma
Tomado de [72]

Editor de gráficos vectoriales utilizado principalmente para diseño gráfico y prototipado. Posee iconos y plantillas listas para usar libremente en los proyectos, puestas a disposición por una comunidad que apoya a la plataforma.

6. Arquitectura

El proyecto es una **DApp** [**DApp**], que utiliza un sistema centralizado para mejorar varias funcionalidades. Los usuarios participan en ella a través de la GUI. Por lo tanto, la aplicación queda claramente dividida en Front-End y Back-End. El Front es el encargado de establecer la comunicación e interacción con los usuarios, mientras que el Back contiene toda la lógica. Como ya se ha mencionado anteriormente, el Back tiene dos partes diferenciadas, una centralizada y otra descentralizada.

Para llevar a cabo el desarrollo de la aplicación se ha seguido el estilo de arquitectura de *Modelo Vista Controlador* ***(MVC)*** [73] donde:

- La *Vista* es el Front-End.
- El *Controlador* engloba al servidor y las funciones recogidas en el smart contract.
- El *Modelo* está compuesto por los esquemas de Mongoose y la estructura de los datos almacenados en el storage del smart contract o en IPFS.

En la siguiente imagen se muestra el contenido de cada capa, así como las conexiones que se establecen entre ellas. Los logos representan las tecnologías utilizadas. A continuación detallamos todas estas comunicaciones.

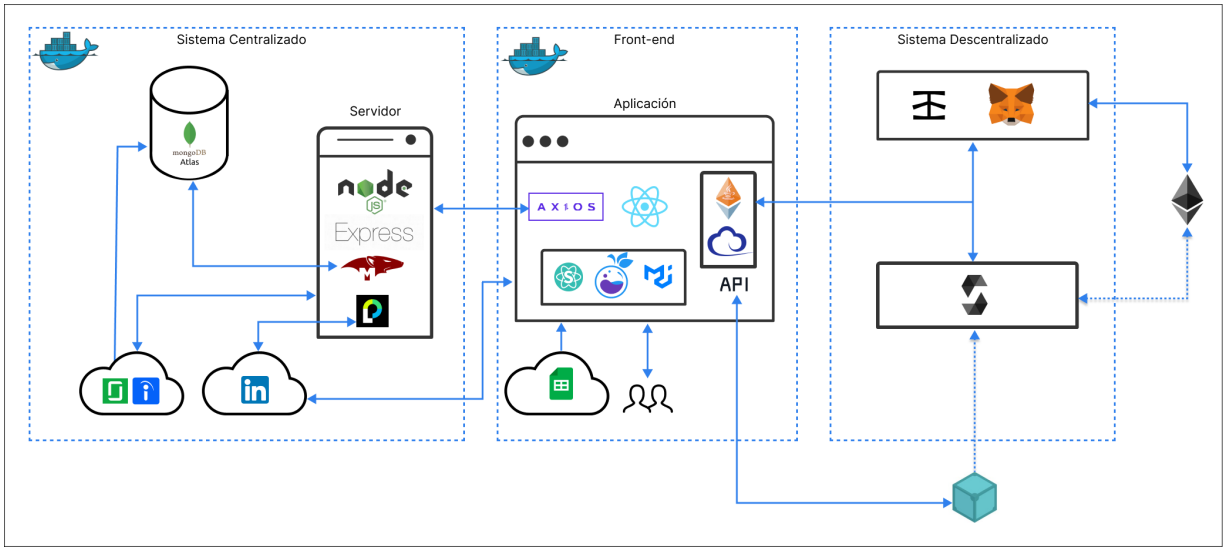


Figura 41: Arquitectura de la aplicación
(Para ver en más detalle la figura, véase Apéndice A)
Fuente propia

6.1. Arquitectura Back-End

El proyecto está basado en un sistema descentralizado para almacenar las denuncias y generar un valor reputacional a partir de ellas. El sistema centralizado se encarga de la verificación con

LinkedIn necesaria para denunciar; de realizar y almacenar el scraping de las reputaciones externas y datos globales de las denuncias para mostrar en las gráficas. Por lo tanto, ambas capas se conectan, a través del Front, a la hora de visualizar las reputaciones y las gráficas.

6.1.1. Arquitectura del Sistema Descentralizado

La arquitectura descentralizada de la DApp está basada en dos tecnologías: Ethereum e IPFS. La primera es la red en la que se ejecutan los 2 smart contracts encargados de toda la lógica de las denuncias y la reputación asociada a ellas. La segunda es un sistema de ficheros descentralizado que permite almacenar los datos de las denuncias de forma gratuita.

Ethereum

El contrato principal almacena, para cada denuncia, una información sobre el fichero de IPFS que la contiene. Además, gestiona la relación entre las denuncias y la reputación de cada empresa. Para ello, se utilizan tokens reputacionales, no transferibles. La lógica de los tokens está implementada en un segundo contrato, siguiendo los estándares de la comunidad de Ethereum, denominados **ERC** [74]. Desde el Front solo se necesita acceso al principal, pues este es el encargado de comunicarse con el de los tokens. En el capítulo de *Implementación* se detallan todas las conexiones entre el Front, los contratos e IPFS.

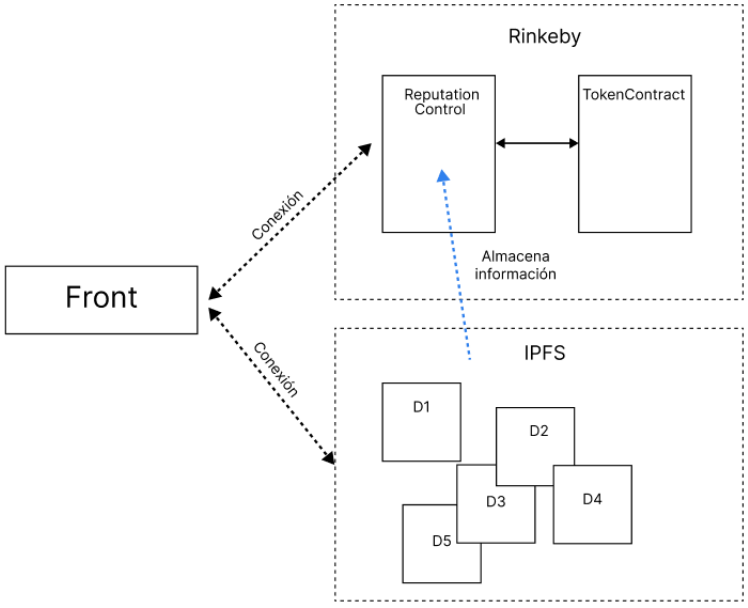


Figura 42: Comunicación Front-Ethereum-IPFS
Fuente propia

Cualquier funcionalidad que se ejecute sobre un smart contract cuesta dinero. En nuestro caso, un ejemplo sería añadir una denuncia al sistema. Se abren dos opciones: los usuarios pagan las denuncias que emiten o el equipo crea una cuenta que financie todos los costes de la aplicación. Si los usuarios las costean, seguramente se reduzca el porcentaje de denuncias falsas. Sin embargo,

esto no incentiva en absoluto a las víctimas. Además, para poder pagar la transacción encargada de añadir una denuncia al contrato, el denunciante debe tener una cartera con ethers, como por ejemplo *Metamask* [75]. Esta es una restricción severa de nuestra aplicación, pues actualmente la mayor parte de las personas no tienen ethers. Por lo tanto, como se pretende desarrollar una aplicación para todos los trabajadores, abierta y sencilla de utilizar, que incentive las denuncias en las empresas y que acople de la mejor manera posible la tecnología blockchain, se ha optado por la segunda opción.

Para acceder desde el Front al contrato principal, almacenado en una red de pruebas de Ethereum, se utilizan dos librerías de forma simultánea. Las librerías en cuestión son **web3j** [76] y **ethers** [77]. La mayoría de DApps incorporan únicamente web3j, pero actualmente algunas de sus funciones están quedando obsoletas y en el futuro parece que se utilizará más ethers. Bastaría emplear una de ellas, pero se han añadido ambas con el objetivo de darle mayor riqueza al proyecto.

Sin embargo, no es suficiente únicamente con las librerías mencionadas. Estas requieren de un **provider** [78], encargado de establecer la conexión con un portal de entrada a la red de Ethereum. En la aplicación se utilizan dos tipos de providers: uno inyectado directamente por Metamask, que se utiliza únicamente para donar; y otro basado en el protocolo HTTP, que permite firmar las transacciones por adelantado y se emplea en el resto de funcionalidades. De este último tipo de provider aparecen dos ejemplos en la aplicación, uno de la librería *truffle* y otro de *web3j*. Por otra parte, como ya se ha mencionado en el estado del arte, Ethereum en realidad es una red distribuida de nodos o clientes [79], encargados de ejecutar los contratos inteligentes. Por lo tanto, existen dos posibles portales de entrada: crear un nuevo nodo o utilizar los que ya existen.

El principal problema de lanzar un nodo a Ethereum desde una de nuestras máquinas es el coste económico y computacional que ello supone. Como la intención de la aplicación no es ganar dinero a través del minado, tiene mucho más sentido utilizar los servicios que ofrecen otras empresas con nodos en la red. Una muy conocida es **Infura** [80], que a parte de ser gratuita, ofrece una API intuitiva con múltiples funcionalidades para la conexión entre las librerías y la blockchain. Pone a disposición de los desarrolladores un cluster de nodos, no uno solo. Se encarga del mantenimiento y el buen funcionamiento. Así, en caso de que uno falle el usuario no se verá perjudicado, cosa que no sucedería en caso de haber desplegado nosotros el nodo. Por todo ello, ha sido la opción elegida.

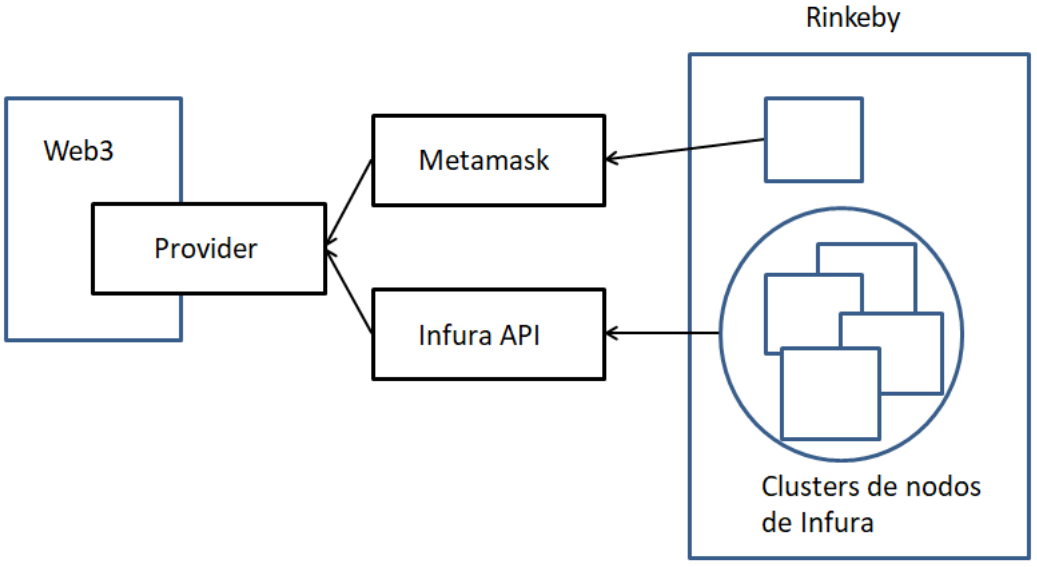


Figura 43: Arquitectura web3j y providers
Fuente propia

Como se ha mencionado antes, nosotros seremos los encargados de financiar todas las denuncias. Utilizamos Metamask, una extensión del navegador que actúa como cartera. Los usuarios que dispongan de ella podrán contribuir en la aplicación, a través de donaciones que se destinarán exclusivamente al pago de las denuncias. Metamask inyecta directamente su propio provider en el Front-End.

Una vez se ha inyectado el provider a web3, la aplicación está lista para comunicarse con el Smart Contract, que reside en la blockchain. Este en realidad no es más que un bytecode. Por lo tanto, para ejecutar sus funciones se necesita una traducción de los nombres y argumentos a bytecode. De igual forma es necesario convertir los bytes que devuelve una llamada. Esto se consigue con el ABI (Application Binary Interface) [81] del contrato. Utilizamos la librería web3 para generar una instancia local de *ReputationControl*, a partir de su ABI. A través de esta instancia se ejecutan las llamadas al contrato de la red.

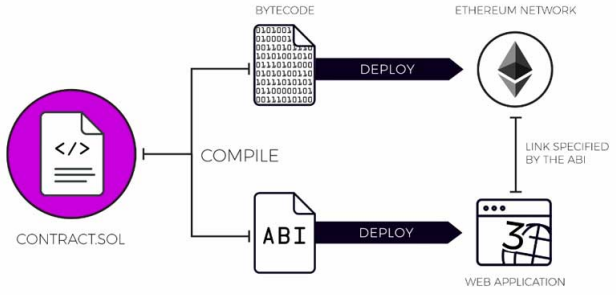


Figura 44: Diagrama de compilación y despliegue de smart contract
Tomado de [82]

IPFS

Almacenar todo el contenido de las denuncias en la blockchain supondría un enorme gasto económico. IPFS es la tecnología perfecta para solventar estos problemas, pues proporciona un sistema de ficheros descentralizado gratuito. Así conseguimos mejorar en eficiencia y gasto económico. De esta forma, en el Smart Contract almacenamos el hash de cada fichero que contiene el cuerpo de una denuncia. La API de IPFS [83] ofrece funciones para acceder a los ficheros, a través de su hash. Además, se utiliza un endpoint que ofrece Infura como puerta de entrada al sistema de ficheros.

6.1.2. Arquitectura del Sistema Centralizado

El sistema centralizado se ha desarrollado sobre Node.js ya que es una herramienta especialmente útil a la hora de desarrollar en el lado del servidor. Se ha construido entorno a un servidor desarrollado mediante Express.js.

Express [47] permite escuchar sobre un puerto determinado y se comporta como una API con respecto al cliente, haciendo posible el envío y recibimiento de datos mediante los *endpoints* [84] al realizar peticiones HTTP/HTTPS. Es un framework que se va actualizando constantemente gracias a la comunidad de desarrolladores [85] por lo que posee multitud de librerías que lo capacitan para operar con cookies, sesiones e *inicio de sesión* por parte de usuarios, entre otras.

Para poder almacenar los datos que posteriormente serán enviados a la aplicación, se crea una base de datos MongoDB en Mongo Atlas, situada en la nube. De forma que se pueda producir la comunicación entre la base de datos y el servidor se emplea Mongoose, una librería que permite la conexión y da soporte a operaciones con MongoDB. Esta requiere la creación de *esquemas* [86] los cuáles definen la estructura de un documento y *modelos* [87], que se aplican sobre colecciones e instancian una clase a partir de un esquema.
Es necesario recalcar que Mongoose, a su vez, para ser capaz de comunicarse con la base de datos y devolver al servidor la información necesaria utiliza el driver de MongoDB [88] para Node.js. Este driver actúa como una API asíncrona que habilita la conexión y permite el intercambio de información mediante promesas de JavaScript.

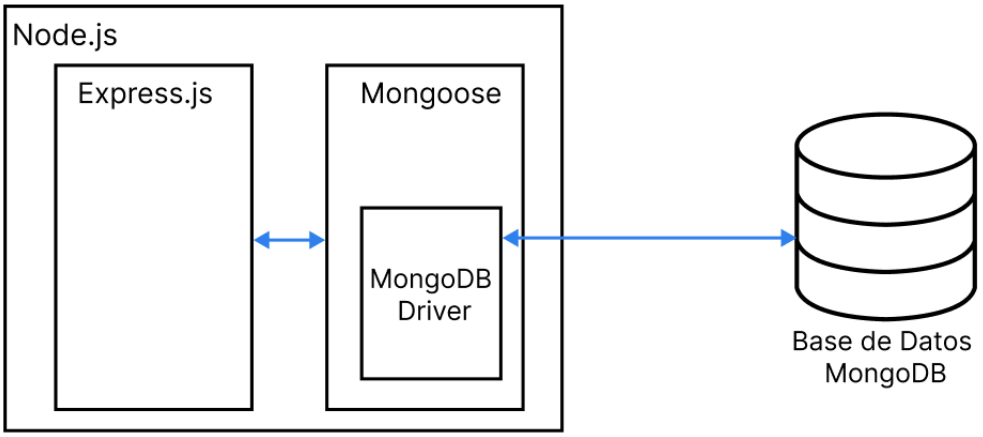


Figura 45: Comunicación Servidor-BD
Fuente propia

El servidor crea una shell de Python que ejecuta un script. Por un lado, envía peticiones a Indeed y Glassdoor que devuelvan el código HTML de las páginas de valoración de distintas empresas para realizar el scraping [89] y por otro, se conecta a Mongo Altas para almacenar los resultados.

A la hora de tratar la autenticación de usuarios, el servidor se conecta con la API de LinkedIn mediante el uso del middleware Passport.js [90]. Se encarga de enviar al usuario a la página de login de LinkedIn mediante una función. Tras realizar el login, los datos son recogidos por Passport para completar la identificación del usuario. En adición, añade a la sesión del cliente el identificador del usuario. Las sesiones se almacenan en el lado del servidor en la base de datos de forma que haya persistencia en caso de fallo.

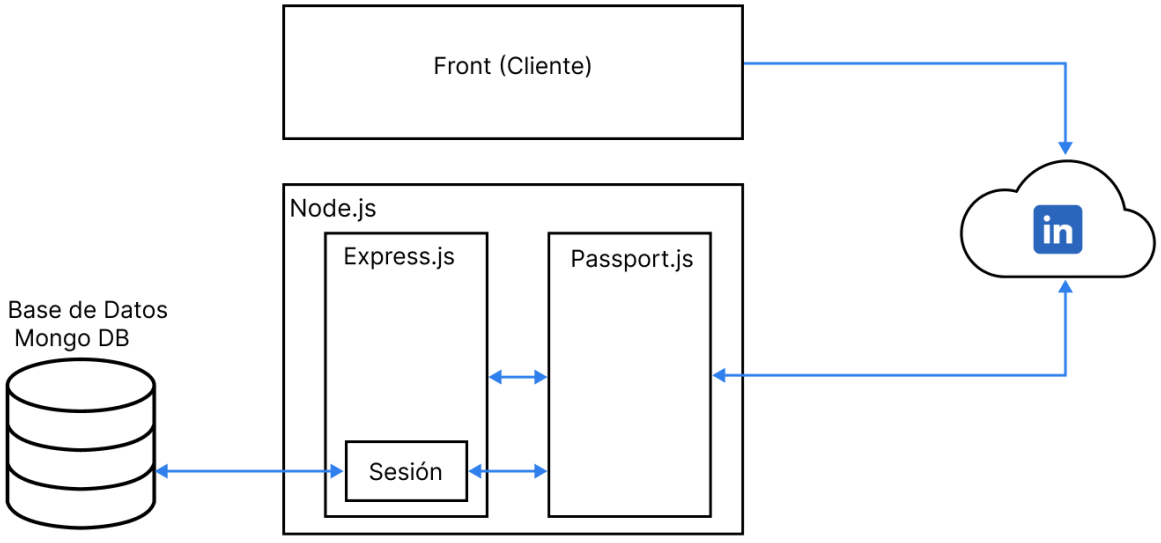


Figura 46: Arquitectura de Passport.js
Fuente propia

Finalmente, el cliente puede comunicarse con el sistema centralizado mediante peticiones GET y POST a las distintas rutas disponibles, siendo estos los que se ocupan de tratar y gestionar los datos de entrada/salida y establecer todas las conexiones internas. Además, el servidor está listo para ejecutarse en contenedores de Docker, dando lugar a un incremento en portabilidad [91] ya que es posible ejecutarlo en cualquier dispositivo y permitiendo una mayor escalabilidad gracias a la ligereza propia de los contenedores de Docker.
Esto da lugar a una arquitectura que escalaría horizontalmente [92] mediante múltiples servidores si se combina con un balanceador de carga, de forma que sería posible atender a una cantidad incrementalmente mayor de peticiones.

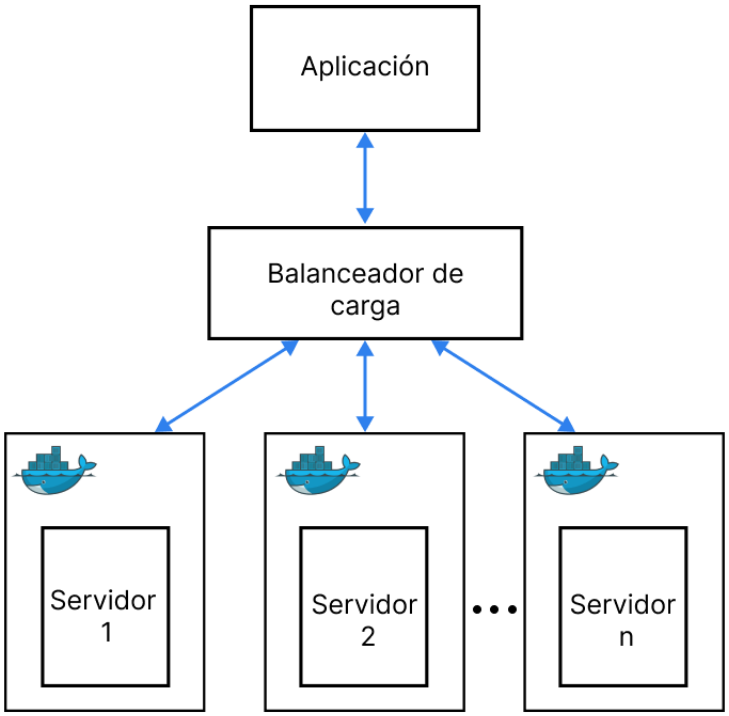
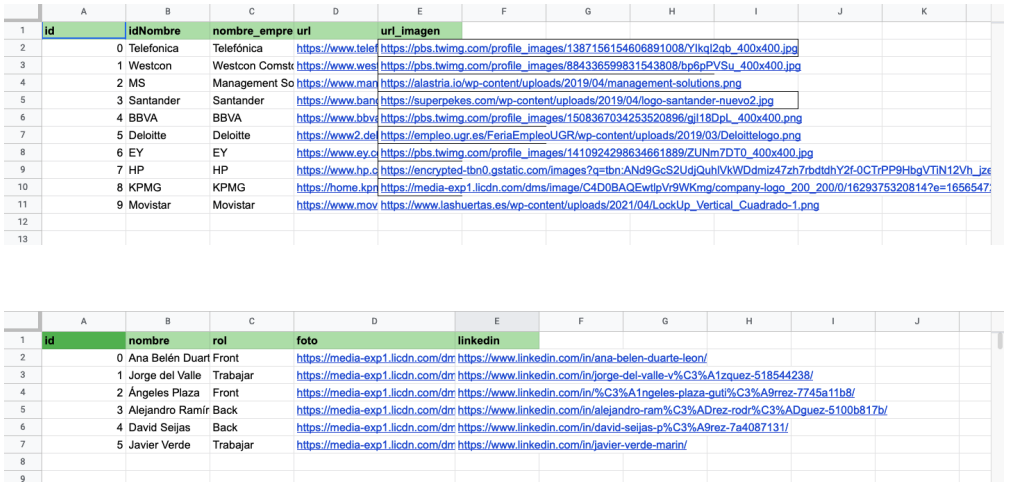


Figura 47: Sistema centralizado multiservidor
Fuente propia

6.2. Arquitectura Front-End

Buscábamos que la forma de añadir nuevas empresas a la plataforma fuese rápida y no incluyese tener que hacer cambios en el código. Asimismo, veíamos innecesario guardar junto con el código las imágenes correspondientes a todas las empresas y miembros del equipo. Por ello, la solución que resolvía ambos problemas fue crear un google sheet en el que pudiésemos guardar no solo la url de las imágenes que necesitábamos, sino también información tanto de los miembros del equipo como de las empresas. De esta manera, empleamos google-spreadsheet [93], un API wrapper de Google Sheets para javascript. Este nos permite conectar nuestra hoja de cálculo con nuestra plataforma.

La hoja de cálculo está compuesta por dos sheets, una de ellas se corresponde a los Miembros del Equipo y la otra a las Empresas. En lugar de guardar imágenes, guardamos la url de la web de las empresas y del LinkedIn de cada usuario, así como de sus fotos, entre otros datos.



	id	idNombre	nombre_empre	url	url_imagen
2	0	Telefonica	Telefónica	https://www.telef	https://pbs.twimg.com/profile_images/1387156154606891008/YIkql2qb_400x400.jpg
3	1	Westcon	Westcon Comstr	https://www.wes	https://pbs.twimg.com/profile_images/884336599831543808/bp6pPVSu_400x400.jpg
4	2	MS	Management So	https://www.mar	https://alastria.io/wp-content/uploads/2019/04/management-solutions.png
5	3	Santander	Santander	https://www.ban	https://superpekes.com/wp-content/uploads/2019/04/logo-santander-nuevo2.jpg
6	4	BBVA	BBVA	https://www.bbv	https://pbs.twimg.com/profile_images/1508367034253520896/qjl18DpL_400x400.png
7	5	Deloitte	Deloitte	https://www2.del	https://empleo.ugr.es/FeriaEmpleoUGR/wp-content/uploads/2019/03/Deloittelogo.png
8	6	EY	EY	https://www.ey.c	https://pbs.twimg.com/profile_images/1410924298634661889/ZUNm7DT0_400x400.jpg
9	7	HP	HP	https://www.hp.c	https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS2UdjQuhlVkWDdmiz47zh7rbdtdhY2f-0CTrPP9HbgVTiN12Vh_jz
10	8	KPMG	KPMG	https://home.kpn	https://media-exp1.licdn.com/dms/image/C4D0BAQEwtlpVr9WKmg/company-logo_200_200/0/1629375320814?e=1656547
11	9	Movistar	Movistar	https://www.mov	https://www.lashuertas.es/wp-content/uploads/2021/04/LockUp_Vertical_Cuadrado-1.png

	id	nombre	rol	foto	linkedin
2	0	Ana Belén Duart	Front	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/ana-belen-duarte-leon/
3	1	Jorge del Valle	Trabajar	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/jorge-del-valle-v%C3%A1zquez-518544238/
4	2	Ángeles Plaza	Front	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/%C3%A1ngeles-plaza-guti%C3%A9rrez-7745a11b8/
5	3	Alejandro Ramír	Back	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/alejandro-ram%C3%ADrez-rodr%C3%ADguez-5100b817b/
6	4	David Seijas	Back	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/david-seijas-p%C3%A9rez-7a4087131/
7	5	Javier Verde	Trabajar	https://media-exp1.licdn.com/dm	https://www.linkedin.com/in/javier-verde-marin/

Figura 48: Sheets con información del proyecto
Fuente propia

Antes de realizar la conexión entre el sheet y el código, necesitamos autenticarnos en la Google Cloud Platform [94]. Debemos crear un proyecto con una cuenta de servicio, habilitar el Google Sheets API para que pueda leer y escribir en hojas de cálculo y obtener las credenciales correspondientes que nos servirán a posteriori para la conexión. Estas credenciales van asociadas a un correo que cree la propia plataforma de Google, y al que deberemos dar permisos para poder editar nuestra hoja de cálculo.

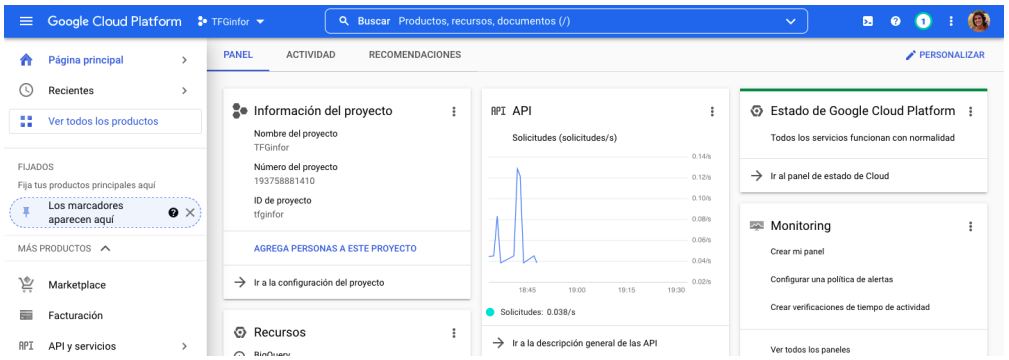


Figura 49: Google Cloud Platform
Fuente propia

Para sacar los datos del Google Sheet y tenerlos disponibles en la plataforma, se hace uso de llamadas específicas en el código con ayuda de las credenciales obtenidas.

Para hacer la comunicación entre el cliente y el servidor posible se ha utilizado Axios, una librería de JavaScript que permite realizar peticiones HTTP/HTTPS asíncronas y obtener respuestas fácilmente procesables a modo de promesa. Además, las distintas librerías que aportan los estilos

y estructura de la aplicación web interactúan con las dos secciones que componen el Back-End mediante hooks de React.

Por último,cabe destacar que el cliente, al igual que el servidor, también tiene la posibilidad ejecutarse a modo de contenedor de Docker. Esto sirve como apoyo a la hora de evitar posibles problemas de compatibilidad que pudieran aparecer durante el desarrollo del proyecto.

7. Implementación

En este capítulo se describe la implementación de nuestra aplicación, analizando las funcionalidades y describiendo los diagramas de secuencias. Por como se gestionó la organización del trabajo y en consenso con los tutores, decidimos elaborar una primera versión funcional, que más tarde daría paso a la versión definitiva. El cambio de la versión inicial a la final lo marcó una reunión que tuvimos con ambos tutores en la que nos aportaron múltiples ideas.

Primero se presenta la implementación de la versión inicial, entrando en detalle de la capa que más se desarrollo: el sistema centralizado. Acto seguido se describe en profundidad la implementación de la versión final, incluyendo diagramas de clases y de secuencias para explicar los contratos y el flujo de las funcionalidades.

7.1. Implementación Versión Inicial

Al comienzo surgieron múltiples dudas de diseño que fuimos resolviendo con el paso del tiempo. Algunas de ellas aparecen en esta versión. Sin embargo, la mayoría se pospusieron para la definitiva. El principal objetivo era obtener un modelo funcional de lo que sería la página web. Gracias a esta versión funcional, fue más sencillo desarrollar la definitiva. Si bien es cierto que tanto la capa descentralizada como el Front son muy simples, la prioridad era familiarizarse con ambas tecnologías.

7.1.1. Sistema Descentralizado

La capa descentralizada se encarga de almacenar las denuncias y gestionar las reputaciones de cada empresa. En esta primera versión, un contrato fue suficiente para el modelo funcional. Tanto los tokens como IPFS se incluyen en la versión definitiva.

Smart Contract

Esta versión inicial no se corresponde con la arquitectura explicada en el capítulo anterior. Existe un único contrato inteligente, sin tokens, encargado de gestionar toda la lógica. Para cada empresa almacena su reputación, una lista de las denuncias y un booleano para saber si está en el sistema. A su vez, una denuncia es una estructura en la que se recoge la mayor parte de los datos introducidos por los denunciantes en el formulario: [1]:

- Id del usuario denunciante
- Texto con la historia de la denuncia
- Fecha de la denuncia
- Tipo de denuncia
- Género del denunciante*
- Relación actual del denunciante con la empresa*

[1] Los campos marcados con * son voluntarios en el formulario.

- Consentimiento para publicar la descripción del suceso en nuestra web

El id del usuario es un string pseudoaleatorio [95] que se obtiene del sistema centralizado. Por lo tanto, la única forma de identificar a una persona que ha puesto una denuncia sería utilizando la API de LinkedIn con las credenciales de nuestra aplicación y conociendo su nombre y contraseña en la red social. Cuando se trabaja con tecnología Blockchain siempre hay que tener en cuenta que todo lo que se carga en ella es público. Esto significa que cualquiera que conozca la dirección del contrato puede ver todos los datos de las denuncias. Consideramos que los campos obligatorios al denunciar son insuficientes para descifrar quien es el denunciante.

A parte de toda la información de las empresas, el contrato guarda una dirección que será el propietario y tendrá permisos exclusivos: añadir empresas y denuncias. También almacena la lista de las empresas, que sirve únicamente para iterar sobre el mapping que contiene toda la información. El Smart Contract se expone en el Anexo, en la sección de Versión Inicial. Se aconseja leerlo, pues los siguientes párrafos detallan la implementación.

El contrato contiene una serie de funciones básicas para obtener la lista de empresas, las reputaciones y denuncias de estas. Además, permite insertar empresas en el caso de que se sumaran a nuestra plataforma. La función más compleja del contrato se encarga de añadir una nueva denuncia al sistema. Es en esta cuando se actualiza la reputación de la empresa. Se mide mediante un entero que comienza en un máximo de 100. Por cada denuncia realizada sobre una compañía, se le resta un punto reputacional. El mínimo es 0, pues en Solidity no existen enteros negativos.

El sistema de reputación tiene varios problemas, que se detallarán en el siguiente capítulo *Implementación de la versión definitiva*. De igual forma, se describen en dicho capítulo las mejoras del contrato, tanto en funcionalidad como en eficiencia y robustez del código.

En caso de conocer como funciona el lenguaje Solidity, recomendamos omitir el siguiente apartado, pues contiene detalles técnicos para que aquellos que nunca han visto este lenguaje puedan entender el código.

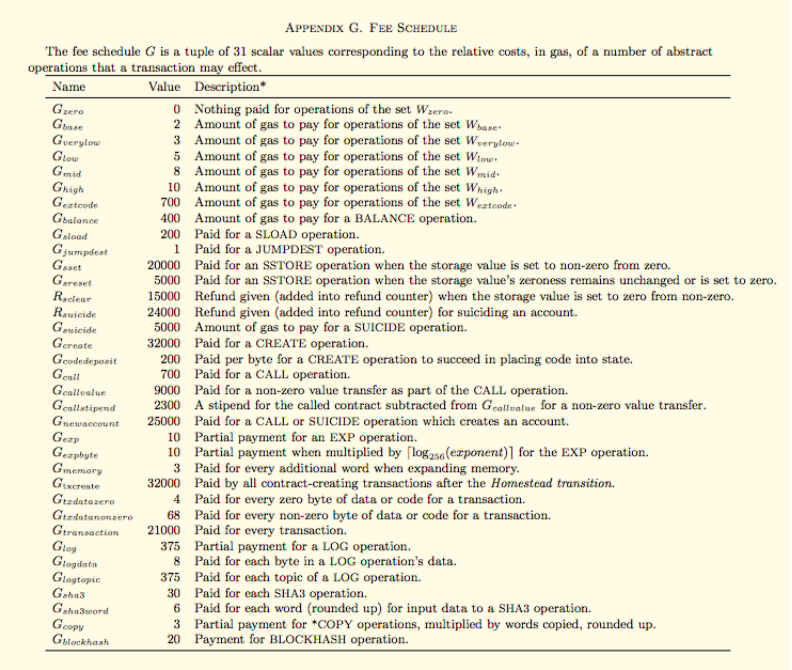
Aspectos técnicos de Solidity [96]

Como se puede observar, este lenguaje es muy parecido a Java, tanto en estructura como en sintaxis. Las dos primeras líneas del fichero hacen referencia a la licencia aplicada, en este caso GPL-3.0, licencia de software libre; y a la versión del compilador. El resto del fichero es el propio smart contract. Los contratos se asemejan en estructura y funcionamiento a las clases de Java. Constan de unos atributos, un constructor y funciones. Los *mapping* son estructuras análogas a un diccionario. En nuestro caso la clave sería el nombre de la empresa y el valor toda la información que se almacena sobre ella. Además, destaca el atributo *owner*, que al ser inmutable solo puede adquirir valor en el constructor. La cuenta que despliegue el contrato será su owner y tendrá permisos exclusivos.

Los *modifier* son líneas de código que se ejecutan al comienzo de cada función a la que se adjunte. Un *require* revierte la ejecución de la función en caso de que no se cumpla la condición. La visibilidad de las funciones es similar a Java. *External* obliga a ser llamada desde el exterior y view indica que los atributos del contrato no se modifican. Existen otras palabras reservadas para el resto de tipos de visibilidades.

Por último, existen tres espacios diferentes en los que se almacenan variables: *storage*, *memory*

y el *heap*. La principal diferencia entre ellos es que el acceso a storage cuesta mucho más gas que en el resto. Storage sería el análogo a un disco duro y las otras dos estructuras sirven para almacenar variables locales, pues son volátiles. Además, memory se utiliza para el paso de parámetros. Los atributos de los contratos, también conocidos como variables de estado se almacenan ahí, por lo que conviene tener cuidado con acceder repetidas veces a ellos si no es necesario.



APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

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

Figura 50: Coste en gas de cada operación
Tomado de [97]

Despliegue del contrato

De entre todas las posibilidades para desplegar un contrato a una red de Ethereum, ya sea de prueba o la principal, destacan dos: desde local o a través de Remix IDE. Remix es la herramienta perfecta para comenzar a explorar Solidity. Sin embargo, tiene sus limitaciones. Por ejemplo, cada vez que cierras el navegador se pierden los ficheros, lo cual es bastante engorroso. Esto hace que Remix solo se utilice para realizar pruebas rápidas y sencillas.

Por lo tanto, se ha optado por la opción de compilar y desplegar el contrato desde local. Para ello, lo primero ha sido preparar el entorno de desarrollo Visual Studio Code, mediante la instalación de la *extensión de Solidity de Juan Blanco* [98]. Una vez implementado el contrato en el fichero *complaintContract.sol*, se compila y despliega a la red de prueba de Rinkeby. Cada acción se lleva a cabo en dos ficheros distintos, *compile.js* y *deploy.js*.

El proceso de compilación consta de varias fases: crear una carpeta de nombre "build.º borrar su contenido en caso de que ya existiera; leer el fichero que contiene el contrato; compilar el código; y añadir un fichero *JSON* con el objeto compilado en la carpeta. Para las etapas de manejos de ficheros se han utilizado las librerías **path** y **fs-extra**. En cuanto a la propia compilación, se realiza a través de la función *compile()* del compilador de Solidity, **solc**. Se ha instalado la última versión

disponible ∧ 0.8.10. Dicha función requiere como parámetro de entrada un JSON. Devuelve otro objeto JSON, con todos los contratos compilados. Se parsea y se recorren todos los contratos (en esta versión solo 1 aunque en la definitiva 2) para almacenarlos en ficheros JSON distintos. A continuación se muestra el fichero *compile.js*

```
const path = require('path'); //rutas de ficheros y carpetas
const solc = require('solc'); //compilador de solidity
const fs = require('fs-extra'); //manejar ficheros y carpetas

const buildPath = path.resolve(__dirname, 'build');
fs.removeSync(buildPath);
const programPath = path.resolve(__dirname,'contracts','complaintContract.sol');
const source = fs.readFileSync(programPath,'utf8');

//Necesario para compilar
const input = {
    language: 'Solidity',
    sources: {
      'complaintContract.sol': {
        content: source
      },
    },
    settings: {
      outputSelection: {
        '*': {
          '*': ['*'],
        },
      },
    },
};

//Compilamos ambos contratos
const output = JSON.parse(solc.compile(JSON.stringify(input))).contracts['
    complaintContract.sol']

fs.ensureDirSync(buildPath);

//output es un diccionario de contratos compilados. La clave es el nombre del
    contrato.
for (let contract in output) {
    //Escribimos un JSON en el path que le mandamos con nombre contract.json
    fs.outputJSONSync(
        path.resolve(buildPath, contract + '.json'),
        output[contract]
    );
}
```

El despliegue del contrato se ha realizado en una red de prueba para no gastar dinero de verdad. Es una práctica común cuando los contratos no son la versión definitiva del proyecto. En nuestro caso, se ha utilizado la red Rinkeby, pues es sencillo conseguir ethers "falsos.[en] ella (link en apartado de Tecnología).

Por otra parte, se necesita el objeto compilado, almacenado en el JSON de la carpeta build. En particular, el ABI y el bytecode, explicado en el capítulo anterior. Para conectar con la red y desplegar el contrato se ha utilizado la librería **web3j**, junto con el provider **HDWalletProvider**. Este requiere la url del nodo de Infura que utilizamos como portal a Rinkeby, y la frase mnemónica

[99] de nuestra cuenta de Metamask (la cuenta paga el coste del despliegue). Ambos datos se almacenan en un .env, pues la frase mnemónica se utiliza para proteger la cartera de Metamask. El provider es diferente al que se usa en el resto de la aplicación y proviene de la librería **truffle**, muy frecuente entre desarrolladores blockchain. En cuanto al propio despliegue, se divide en tres etapas, aunque en el código sea una única línea: crear una instancia del contrato con el ABI; crear la transacción para desplegarlo con *.deploy(BYTECODE)*; y lanzarla con .send({...}). Se especifica el consumo máximo de gas para desplegar el contrato dentro del send. A continuación se muestra el código del fichero *deploy.js*

```
1  const HDWalletProvider = require('@truffle/hdwallet-provider');
2  const Web3 = require('web3');
3  require('dotevn').config();
4  const compiledContract = require('./build/complaintContract.json');
5
6  const provider = new HDWalletProvider(
7    process.env.SEED_PHRASE, //Frase mneum nica
8    process.env.INFURA_URL // URL del nodo de INFURA
9  );
10
11 const web3 = new Web3(provider);
12
13 const deploy = async () => {
14   const accounts = await web3.eth.getAccounts();
15   console.log('Attempting to deploy from account', accounts[0]);
16   const result = await new web3.eth.Contract(compiledContract.abi)
17     .deploy({ data: compiledContract.evm.bytecode.object})
18     .send({ gas: '10000000', from: accounts[0] });
19
20   console.log('Contract deployed to', result.options.address);
21   provider.engine.stop();
22 };
23
24 deploy();
```

Donaciones

La última funcionalidad asociada al sistema descentralizado que se desarrolló en esta primera versión fue la posibilidad de donar. Todos los costes de la aplicación los cubre una cuenta creada por el equipo. Por lo tanto, recaudar fondos es esencial para que el proyecto se extienda en el tiempo. Como ya se ha mencionado en el estado del arte, las DApps suelen financiarse a través de crowfunding. Se ha decidido utilizar este método, aunque el usuario debe tener la extensión de Metamask en el navegador para apoyar económicamente al proyecto. A pesar de ser una limitación al proyecto, consideramos que era la forma más sencilla de aceptar donaciones de forma transparente y rápida.

En cuanto a la implementación de esta funcionalidad, se realiza todo sobre el Front-End. El usuario introduce la cantidad de ethers a donar. En ese momento, si tiene la extensión de Metamask en el navegador, este inyecta en una instancia de web3j el provider necesario para la conexión con Ethereum. A través de las funciones de la librería se envía la transacción, cuyo destinatario es la dirección del contrato ReputationControl. El usuario debe confirmar el envío de la transacción en Metamask. Le salta instantáneamente un pop-up. Además, puede ver que la cantidad donada ha aumentado en la página web, comprobando que la aplicación opera con la mayor transparencia.

7.1.2. Sistema Centralizado

El caso del sistema centralizado es totalmente diferente. En esta parte del proyecto se pudo desarrollar un modelo detallado, totalmente funcional. No obstante, no se completaría su implementación definitiva hasta la versión final.

En la primera versión, la meta del sistema centralizado era la de administrar el inicio de sesión de usuarios. Se llegó a la conclusión de que los usuarios se podrían identificar mediante su cuenta de LinkedIn ya que al ser una red social orientada al uso laboral sería más cercana a los usuarios para los que va dirigida la plataforma. Además, de esa manera se almacenarían la menor cantidad de información de los usuarios posible.

Para poder habilitar el login con LinkedIn se necesitaba, en primer lugar, acceso a la API **[LinkedIn_app]** con motivo de hacer peticiones sobre usuarios. Se creó una Aplicación en LinkedIn Developers y se añadió como producto "Sign In with LinkedIn"para poder utilizar las funcionalidades de login. A su vez, se obtuvieron unas claves de autenticación (ID de cliente y Clave privada) para usar en la aplicación a la hora de realizar las peticiones.

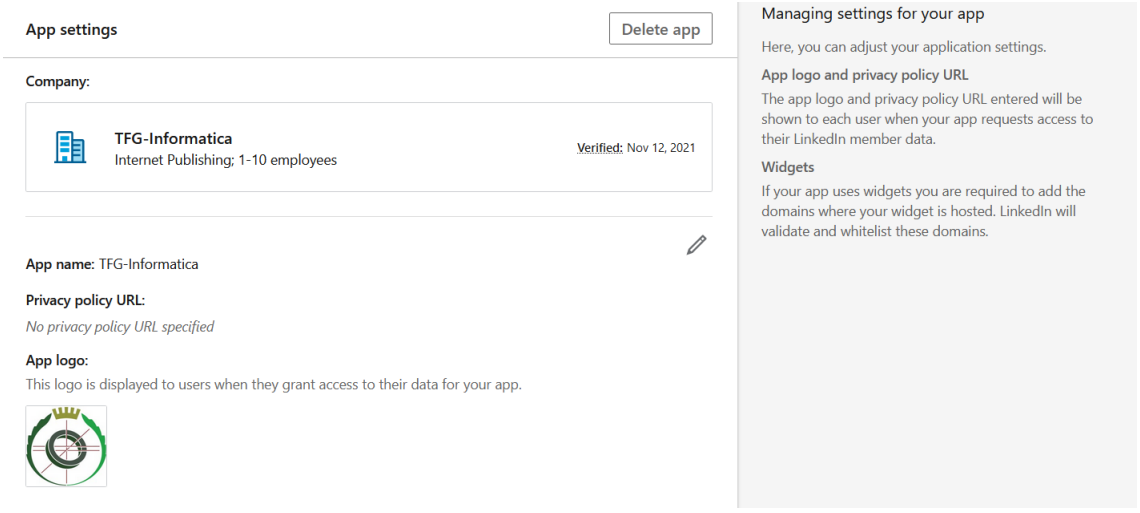


Figura 51: Panel de aplicación de LinkedIn Developers
Fuente propia

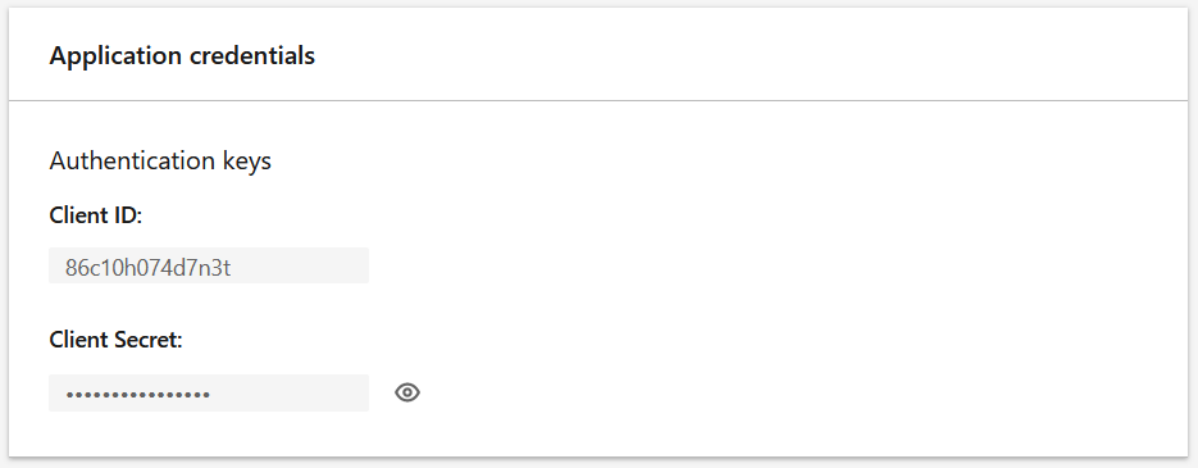


Figura 52: Credenciales de la API de LinkedIn
Fuente propia

El servidor

El desarrollo del sistema centralizado se inició creando un servidor mediante Express, sobre el cuál se desarrolla el resto de componentes y se configuró para escuche sobre el puerto 3001. Antes de incluir otras tecnologías se instalaron e incorporaron algunos middlewares necesarios para tratar las peticiones como *dotenv* [100] y *cors* [101], el cuál fue configurado para que se aceptaran peticiones de la aplicación ya que la comunicación cliente-servidor es cross-site. En adición, se creó un .env [102] donde se almacenan distintos datos secretos como las claves obtenidas al crear la aplicación.

La base de datos

Es necesario poder almacenar algún tipo de identificación para que se pueda efectuar un login satisfactorio, por lo que para ello se eligió utilizar Mongo Atlas [103]. Este servicio ofrece una base de datos en la nube fácilmente consultable y que aprovecha la eficiencia de MongoDB [104] además de aportar trazabilidad de las peticiones que recibe.

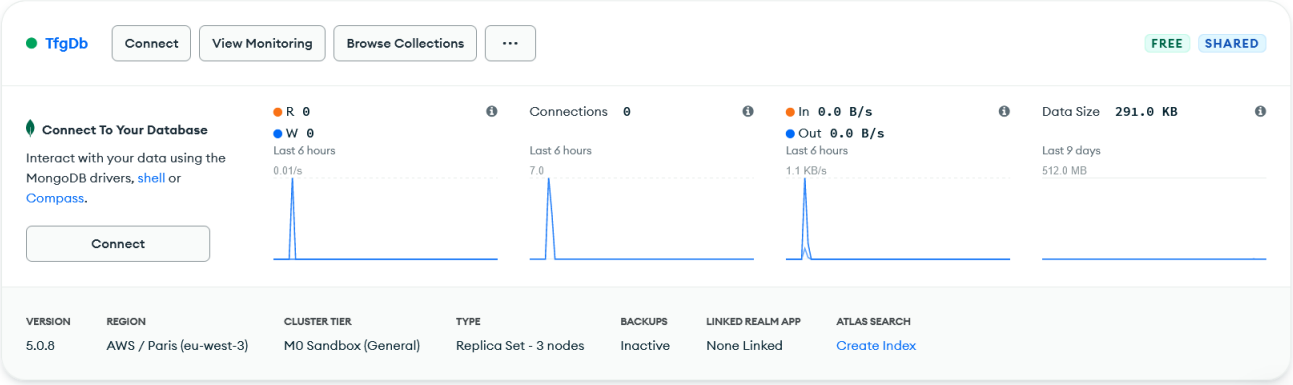


Figura 53: Base de datos Mongo Atlas
Fuente propia

A la hora de desarrollar el formato de los documentos que representan a cada usuario, se partió de la información que se podía obtener a partir de la API de LinkedIn. En el caso de este proyecto solo se puede utilizar el *Lite Profile* [105], el perfil con menos permisos e información disponible aportado por LinkedIn ya que no nos encontramos dentro del programa de partnership.

El Lite Profile contiene los siguientes datos:

- ID de miembro.
- Nombre.
- Apellido.
- Foto de perfil.

Debido a que se desea que los usuarios posean el mayor anonimato posible, se descartan nombre, apellido y foto de perfil y se crea el formato del documento de usuario a partir del ID de miembro.

El ID proviene de un *URN* que identifica los recursos u objetos de una web y que está estructurado según el esquema urn [106]:

urn:{namespace}:{entityType}:{id}

El tipo de URN utilizado se denomina Person URN [107], cuya función es la de identificar a un usuario de LinkedIn en una aplicación y cuyo formato, basado en el esquema anterior es:

urn:li:person:{id}

El namespace y entityType que definen que es un usuario de LinkedIn son comunes a todos los URNs de tipo Person independientemente de la aplicación. No obstante, el identificador [108] es exclusivo para nuestra aplicación, por lo que no tendrá uso en cualquier otra que no utilice las mismas credenciales.

Por otro lado, a cada nuevo usuario el servidor le asigna un nombre pseudoaleatorio de 15 cifras y/o letras que no haya sido utilizado aún mediante el módulo *randomstring* [109]. La razón por la que se usa este método es para mantener el anonimato de cara a las denuncias ya que al introducirlo un usuario, podría dejar pistas sobre su identidad de forma inconsciente [110].
De esta forma, el documento utilizado para identificar al usuario y que se almacena en la base de datos está compuesto por su ID en la aplicación obtenido mediante la API de LinkedIn y el nombre de usuario generado. En caso de vulnerarse la base de datos, únicamente con los datos almacenados sería imposible conocer al usuario que representan o de donde provienen los mismos por lo que la propia estructura también refuerza la protección al denunciante y su derecho a la privacidad.

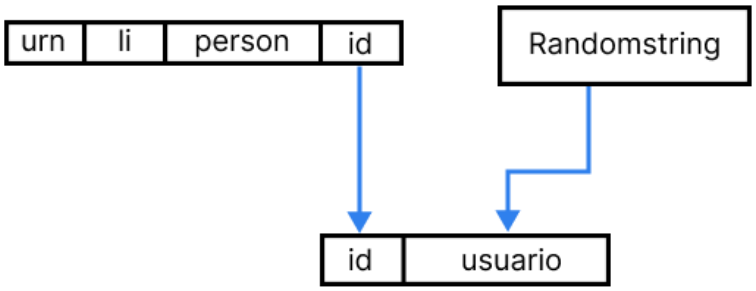


Figura 54: Formato de identificación de usuario
Fuente propia

Como se describió en el apartado de arquitectura, se utiliza la librería Mongoose para que el servidor pueda conectarse a la base de datos. Se incluyeron los componentes de la URI que apunta a la base de datos en el .env y se realizó la conexión con la base de datos mediante el método *connect* de Mongoose. Para poder realizar el intercambio de información e introducir datos se debe crear un esquema que indique el formato de estos datos y un modelo que aplique el esquema [86]. El modelo [87] es la interfaz que hace posible la comunicación entre servidor y base de datos.

En el siguiente fragmento de código se muestra la creación del modelo de Mongoose:

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  const UserSchema = new Schema({
5    l_id :{ type : String, required : true, unique : true},
6    u_name : { type : String, required : true, unique : true}
7
8  }, { collection : 'userInfo' });
9
10 const User = new mongoose.model('userInfo', UserSchema);
11 module.exports = User;
```

Se define un esquema *UserSchema*, que indica que el documento posee un campo *l_id* que corresponde con el ID y un campo *u_name* que contiene el nombre de usuario generado. En adición a la definición de la estructura también se indica la colección sobre la que se almacenarán los documentos de este tipo, *userInfo*. El modelo *User* puede ser utilizado por el servidor y permite realizar consultas a la base de datos e insertar nuevos documentos mediantes new User.

Autenticación de usuarios

Una vez el servidor y la base de datos estaban conectados y funcionales se inició el proceso de desarrollo de la autenticación de usuarios.

En primer lugar se hizo uso de *express-session* [111], un middleware que permite crear y administrar sesiones. Se envía una cookie llamada *connect.sid* al cliente con el identificador de la sesión mientras que la información se almacena en el servidor para poder obtener la información de la sesión posteriormente mediante el identificador en caso de que se produzcan futuras peticiones.

A la hora de implementar el manejo de la sesión se definió el secreto de la sesión, la fecha de expiración y el se establece la cookie a modo httpOnly [112] para que no se pueda ver su contenido mediante document.cookie e intentar prevenir ataques de tipo Cross-site Scripting (XSS) [113]. En un despliegue real también se utilizaría el modo secure para que solo pueda enviarse a través de conexiones HTTPS.

Tras configurar el middleware del express-session se realizó la configuración del middleware *Passport.js*, que permite utilizar distintas estrategias dependiendo del tipo de autenticación que se quiera realizar. La estrategia empleada se basa en *OAuth 2* [114], un framework de autorización que permite delegar en otro servicio o plataforma como LinkedIn la verificación del usuario así como obtener recursos del usuario en esa plataforma. En primer lugar se inicializa el middleware mediante *passport.initialize()* y se realiza *passport.session()* [115] de forma que cuando se autentique el usuario, se modifique el valor de *user* en la sesión.
Posteriormente es necesario crear una ruta, en nuestro caso */auth/linkedin* [116], que se utiliza cuando el cliente pulsa al botón de inicio de sesión. Dentro de esta ruta se llama a *passport.authenticate()* que redirige a LinkedIn. Una vez el usuario escribe su usuario y contraseña y pulsa a iniciar sesión se vuelve a la aplicación.

Este proyecto utiliza la estrategia *LinkedInStrategy* [116], la cuál necesita las credenciales de API obtenidas al crear nuestra aplicación en LinkedIn Developers, una URI de redirección que se llama cuando se retorna desde LinkedIn y el scope, que indica el tipo de perfil que se va a obtener y que en nuestro caso será el Lite Profile. Además posee una función de callback cuyo fin es el de realizar las acciones necesarias con los datos aportados por el perfil.

Una vez se realiza el login en LinkedIn se envía un código a modo de parámetro de query a la ruta definida en la URI de redirección */auth/linkedin/callback* En esta ruta también se realiza passport.authenticate() [117]. pero se utiliza para comprobar que el usuario haya confirmado el inicio de sesión de forma efectiva y activa la estrategia la cuál envía el código obtenido previamente a la API de LinkedIn para recibir el perfil. Posteriormente, se llama a la función de callback recogida en la estrategia de Passport y que se ocupa de tratar con el perfil para llevar a cabo nuestra autenticación.

A continuación se detalla la función de callback que realiza nuestra estrategia de LinkedIn:

```
1  function(accessToken, refreshToken, profile, done) { //Callback
2    //Verificacion asincrona
3    process.nextTick(function () {
4      //vamos a comprobar si el usuario esta en la BD, si no, se aniade
5      User.findOne({'l_id' : profile.id}, (err, doc) =>{
6        if(err) return done(err, null);
7        else if(!doc){
8            let unique = false;
9            let username;
10           /*
11             Generamos un nombre de usuario y comprobamos que no exista ya:
                   Normalmente el bucle tendra una
12           complejidad O(log n) ya que es dificil que se repita un nombre de
                   usuario de tal longitud y se posee un indice en l_id.
13           */
14           while(!unique){
15             username = genName.generate({length:15,charset: 'alphanumeric'});
16             unique = User.findOne({'u_name': username}, (err, res) =>{
17               if(err) return false;
```

```
18                else if(!res) return true;
19                else return false;
20              });
21            }
22            //Guardamos el id y el usuario generado de forma aleatoria
23            const new_user = new User({ l_id : profile.id , u_name : username});
24            new_user.save(function(){});
25
26            //Se utiliza el documento del nuevo usuario
27            return done(null,new_user);
28        }
29        else{
30            //Si el usuario ya existe se devuelve el documento
31            return done(null, doc);
32        }
33      });
34    });
35  }
```

La función recibe el perfil y se busca su id en la base de datos. En caso de que no esté, se crea un nuevo nombre de usuario pseudoaleatorio hasta que este no exista en la base de datos. Se debe tener en cuenta que es altamente difícil que se repita ya que el conjunto de nombres de usuario que se pueden generar se basa en una permutación con repetición [118] de 36 elementos totales (números y letras) y 15 utilizados en el momento. Esto implica que incluso con una gran cantidad de usuarios registrados, el bucle de creación de nombre y búsqueda de este en la base de datos solo se atravesará una vez.

Una vez se tiene un nombre de usuario libre, se introduce el nuevo usuario en la base de datos mediante el modelo detallado anteriormente y se envía en la función *done()*. Si, por otro lado, ya existe un usuario con ese id, se envía el documento que contiene su información directamente.

La llamada a done() [117] conlleva la ejecución de la función *serializeUser()*, la cuál extrae el id de Objeto del documento de MongoDB, distinto al id de LinkedIn y se almacena en la sesión que se envía al usuario. De esta manera, al recibir una petición por parte del usuario se llama a *deserializeUser()*, que se ocupa de buscar al usuario mediante el id de Objeto en la base de datos y se devuelve el nombre de usuario, almacenándolo en *req.user*.

Login y logout

El Front recibe el nombre de usuario mediante la ruta */getuser* cuyo fin es el de enviar el contenido almacenado en req.user. Para realizar el cierre de sesión se recibe una petición en la ruta */logout*, que se ocupa de destruir la sesión y limpiar el valor de req.user.

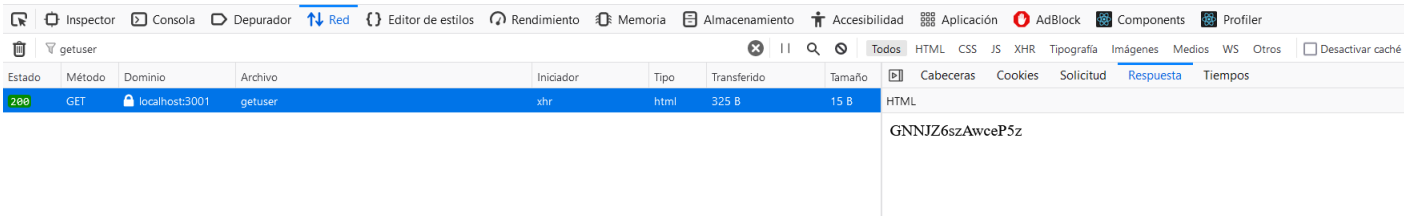


Figura 55: Solicitud de nombre de usuario
Fuente propia

En la siguiente figura se muestra de forma esquematizada el proceso de autenticación mediante Passport.js y obtención del nombre de usuario:

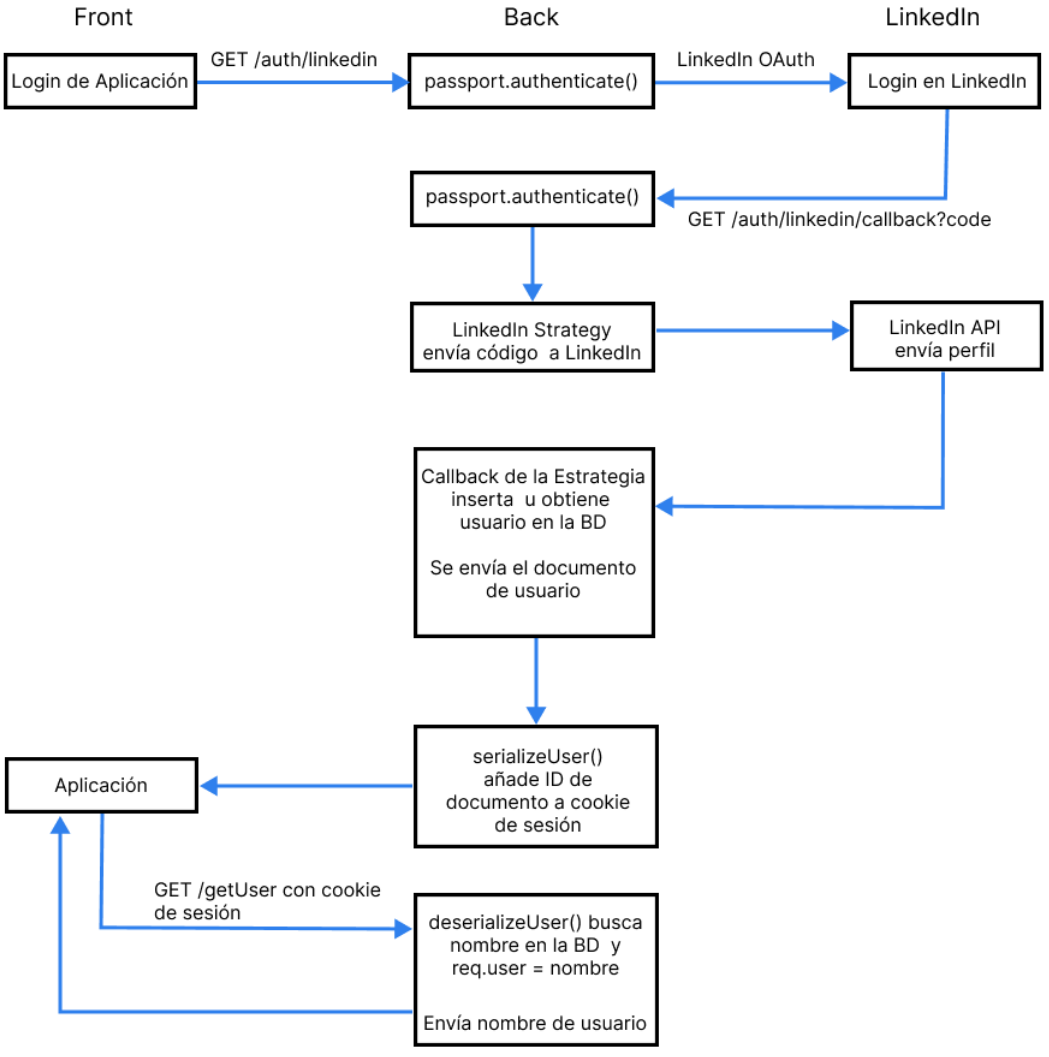


Figura 56: Diagrama de funcionamiento de Passport.js con LinkedIn
Fuente propia

7.1.3. Front-End

En una primera versión, el subequipo de Front-End se centró sobre todo en crear componentes que recogieran la funcionalidad principal de la plataforma y que funcionasen. Se optó por no utilizar plantilla y crear un diseño de la plataforma desde cero. Por ello, el primer paso dado fue diseñar un Mockup de cómo serían las distintas pantallas usando Balsamiq. Este Mockup se fue perfeccionando durante unos días, teniendo en cuenta los principios de diseño de interfaces. Este primer prototipo realizado en Balsamiq quedó de la siguiente manera:

En principio, el login tendría su propia pantalla donde aparecería un texto explicando ciertos aspectos sobre las cookies y el tratamiento de los datos que realiza la plataforma. Creíamos conveniente resaltar que no se almacenan datos personales ya que es uno de los fundamentos de los sistemas descentralizados.

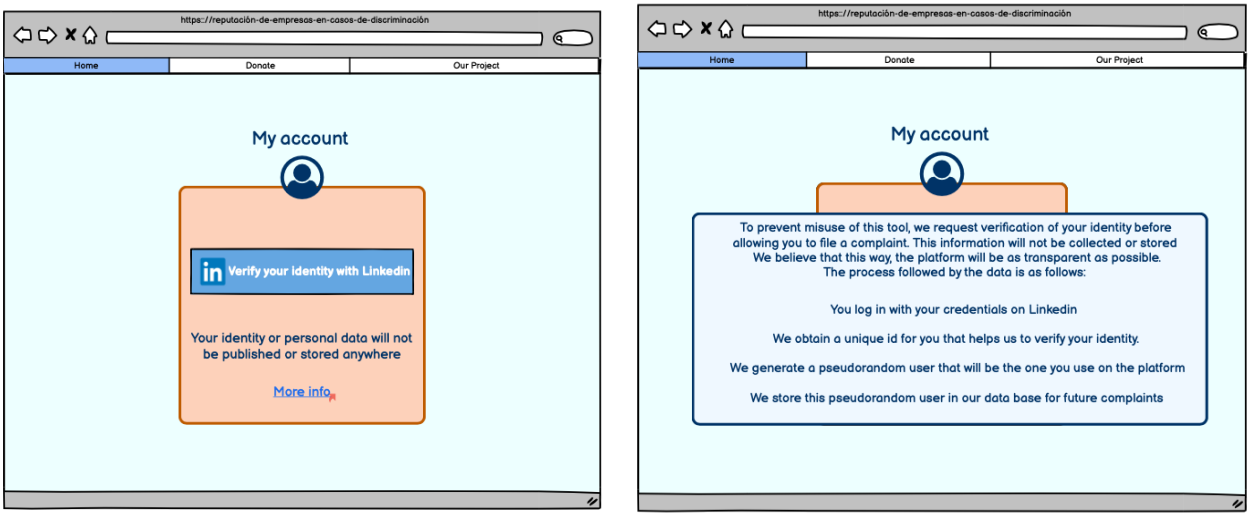


Figura 57: Boceto del login
Fuente propia

Para el diseño de la página de inicio se plantearon dos modelos distintos que en esencia tenían los mismos componentes y funcionalidades pero distribuidos de forma diferente. En esta primera fase de diseño, aún no estaba del todo claro las funcionalidades exactas que incluiría la plataforma, que se han ido modificando a lo largo del diseño. Por ello, en un primer lugar se pensó en incluir un mapa con lugares desde donde se había usado la plataforma para denunciar pero se acabó acordando que esto carecía de sentido además de que en cierto aspecto "desanonimizaba"las denuncias interpuestas por los usuarios.

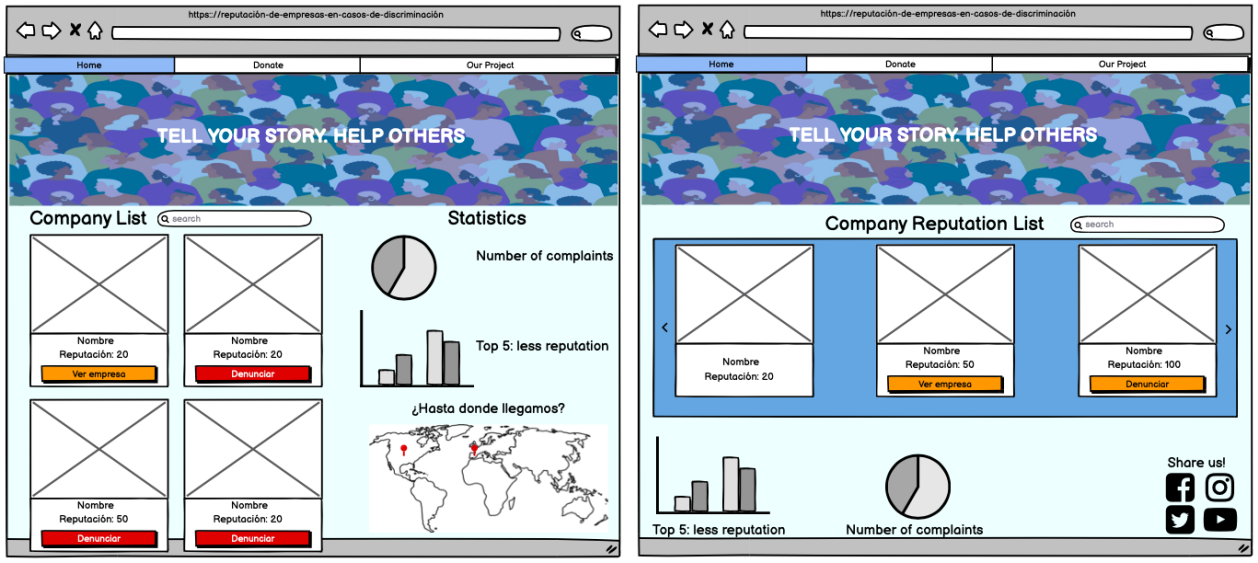


Figura 58: Boceto de la página principal
Fuente propia

Se podría acceder a cada una de las empresas incluidas en la plataforma para visualizar las denuncias públicas hechas contra ésta, así como gráficas con sus estadísticas y su reputación calculada en función de una serie de métricas. Cada una de las pantallas pertenecientes a una empresa tendría el aspecto siguiente:

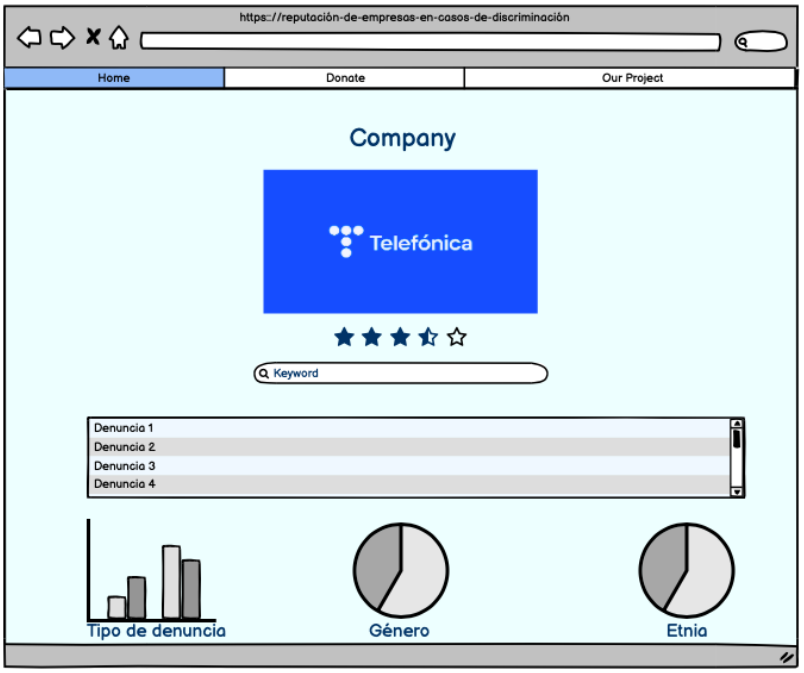


Figura 59: Boceto de la página de una empresa
Fuente propia

Para describir el proyecto y presentar a cada uno de los integrantes del equipo se diseño una pantalla que se presenta a continuación. Además, otra pantalla animaría a los usuarios a donar para que el coste de enviar una denuncia a la Blockchain corriese por parte del propio smart contract de la plataforma y no por parte de la víctima.

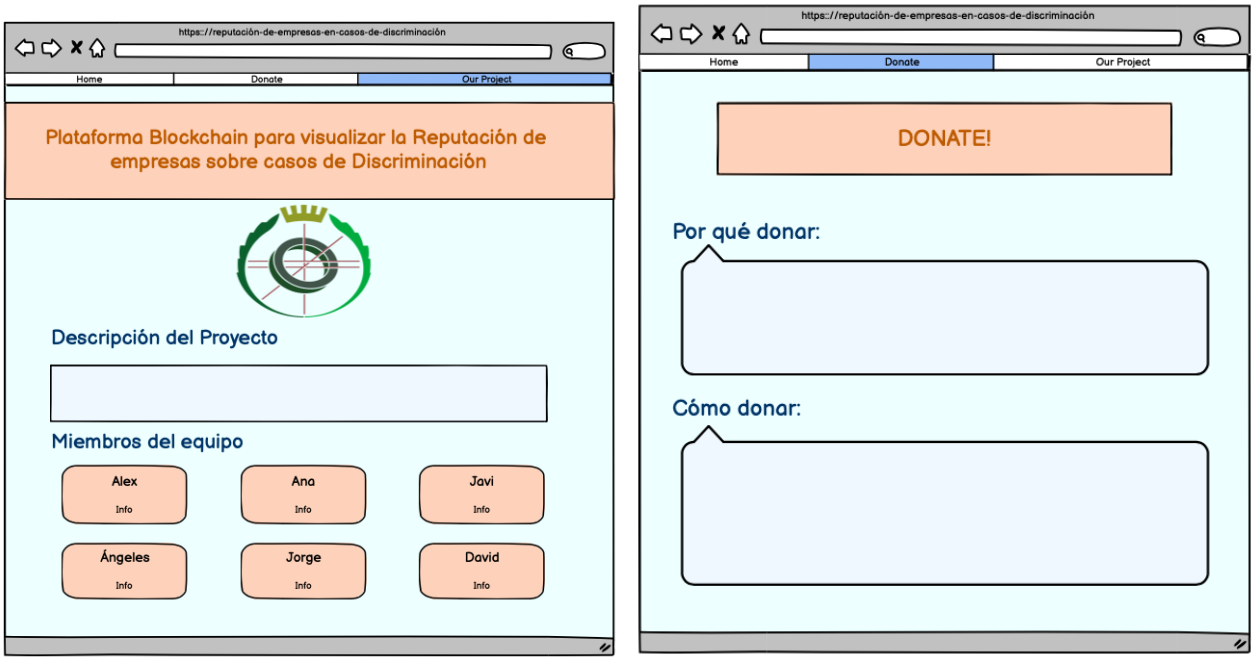


Figura 60: Boceto de la página de información
Fuente propia

Debido a ciertas limitaciones en cuanto a tiempo y conocimiento de los programadores no se pudo recrear exactamente el modelo previamente diseñado con la herramienta de prototipado pero ajustándose lo máximo posible a éste, la primera versión tenía el aspecto de las imágenes que aparecerán a continuación:

- **Home**

Figura 61: Diseño de la página principal
Fuente propia

- **Pantalla para visualizar una empresa**

Esta pantalla tiene tres funcionalidades principales: poder denunciar a la empresa, poder conocer cuáles han sido las denuncias a esta empresa, y visualizar de una forma más general las estadísticas de las denuncias.

El botón para denunciar aparece al comienzo y en el centro de la pantalla, con el objetivo de que sea una acción fácil de encontrar y rápida de realizar.

A la hora de diseñar esta pantalla, teníamos claro que necesitábamos mostrar las denuncias realizadas a las diferentes empresas sin comprometer la información compartida por los denunciantes. Por ello, decidimos emplear una tabla que reflejase únicamente aquellos datos por los que no se pudiese identificar a la persona. Sin embargo, existe un campo adicional para leer la experiencia del usuario, con la que sí se puede llegar a identificar al denunciante. Este campo solo será visible si el usuario lo ha permitido en la denuncia.

Para la tabla elegimos material-table [119]. Primeramente diseñamos una tabla básica, a la que decidimos añadir una serie de funcionalidades que aportaban valor: capacidad de filtrar las denuncias y de ordenarlas acorde a cada una de las columnas. Asimismo, al pulsar el botón de leer la experiencia aparecía un dialog.

Además, para poder visualizar las estadísticas de forma general, decidimos incluir una serie de gráficas con información sacada de las denuncias.

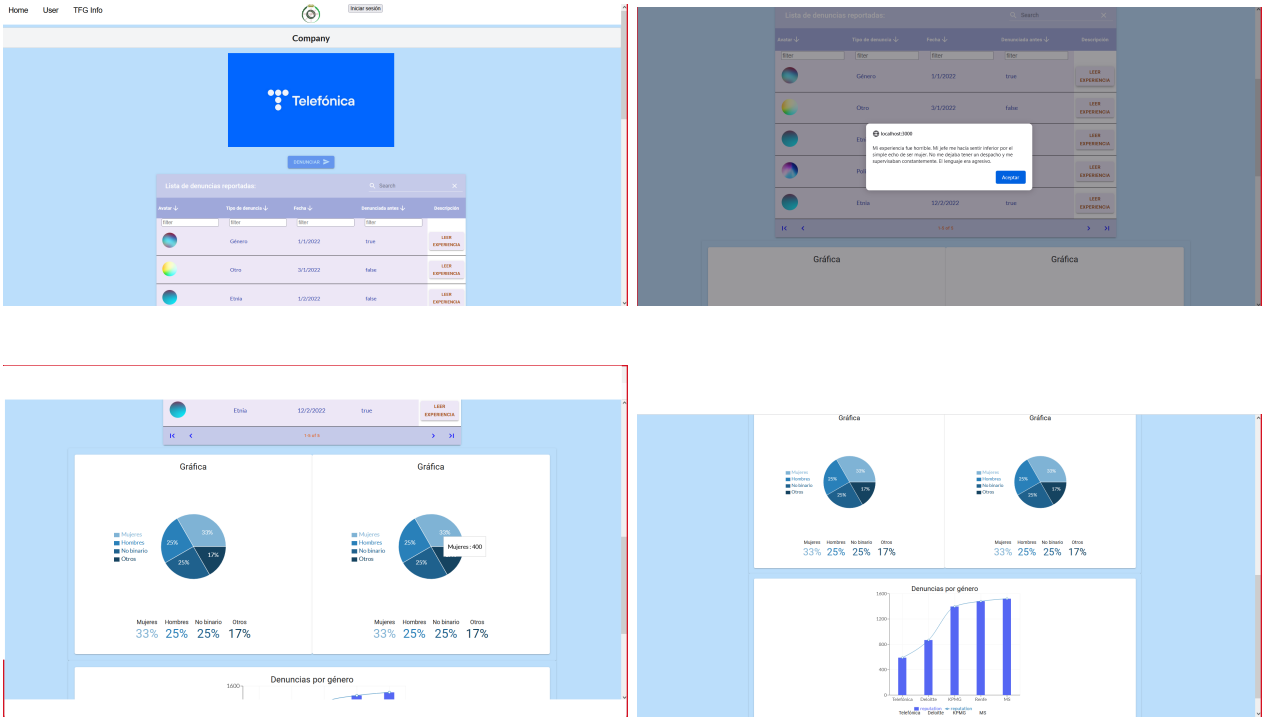


Figura 62: Diseño de la página de una empresa
Fuente propia

- **Pantalla con el Formulario**

Para generar el formulario empleamos *Semantic UI*, ya que era el *framework* que nos ofrecía un diseño más similar al que buscábamos y del que podíamos obtener todas las funcionalidades que necesitábamos.

A la hora de diseñar el formulario, tratamos que el usuario tuviese que realizar el menor esfuerzo a la hora de denunciar y que, a la vez, nosotros pudiésemos recopilar esta información de forma más sencilla. Por ello, en la mayoría de campos, intentamos que las respuestas fueran de opción múltiple, en lugar de que fuese el usuario el que escribiese.

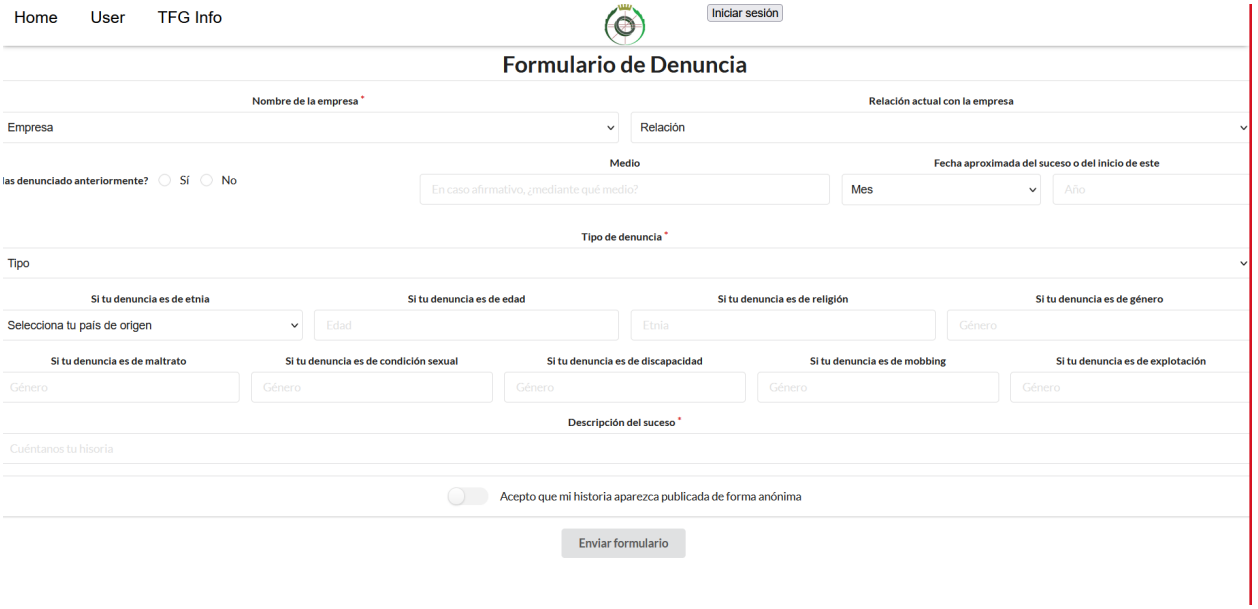


Figura 63: Diseño del formulario
Fuente propia

La lista de tipos de las denuncias fue una puesta en común de todos los miembros, a partir de la literatura existente relativa a denuncias y discriminación en las empresas [120, 121, 122, 123, 124, 125, 126]. Se puede escoger entre: Racismo, Discriminación por Género, Orientación Sexual, Religión, Edad, Discapacidad y Otro. El nombre de la empresa también es un scroll de opciones, para evitar que se introduzcan errores a la hora de escribirlo. Los otros campos en los que se marca la elección es en la relación actual con la empresa y en el país de origen. El resto de campos son espacios de texto para que el usuario escriba libremente lo que desee.

Tan solo son obligatorios el nombre, el tipo y la descripción del suceso. Además, el consentimiento está por defecto a No. En caso de que el usuario quiera aceptar que se muestre la descripción en la tabla de denuncias de una empresa, solo ha de clickear en el deslizante de abajo. El formulario se mejora en la versión definitiva para que el usuario lo entienda mejor.

- **Pantalla con información del proyecto:**

Esta pantalla es aquella que informa al usuario sobre nuestro proyecto, su motivación y la repercusión que espera tener. En ella aparece una breve descripción del proyecto y una tarjeta para identificar a cada uno de los participantes.

Figura 64: Diseño de la página de información
Fuente propia

7.2. Implementación Versión Definitiva

Tras una reunión con nuestros tutores en la que presentamos una demo de la versión inicial quedó claro que había muchas mejoras tanto en el diseño de las pantallas, para proporcionar una mejor experiencia al usuario, como a la hora de introducir nuevas tecnologías que hicieran más eficiente y enriquecedor nuestro proyecto.

7.2.1. Sistema Descentralizado

El contrato de la versión inicial presentaba un claro problema de eficiencia económica, ya que almacenar todos los datos de la denunciar en la blockchain es muy caro. Además, acceder a ellos para mostrar las denuncias o representar las gráficas, encarece aún más el modelo. Para solucionar este problema se introduce una nueva tecnología: **IPFS**. Este sistema de ficheros permite almacenar los datos de cada denuncia sin costes adicionales.

Por otra parte, el sistema de reputaciones inicial es muy básico:

- Impide ganar reputación si la empresa tiene comportamientos ejemplares.
- Tan solo contabilizan las primeras 100 denuncias a una empresa, pues en ese momento la reputación sería 0.
- Valor inicial (100) intuido por los componentes del equipo que puede no ajustarse a la realidad.
- Escasa robustez al comienzo, pues todas las empresas tienen reputación máxima. Puede ser que algunas no merezcan esa puntuación inicial.
- De hecho, si se añade una empresa cuando el volumen de las denuncias es considerable, ésta entra con reputación máxima mientras que otras tendrán reputación muy baja.
- El sistema de reputación no es novedoso pese a incluir blockchain.

Analizando todos estos factores, decidimos introducir tres importantes cambios: incluir **tokens** que midan la reputación en base a las denuncias recibidas, extraer datos de reputaciones de otras páginas webs especializadas en ello y establecer una **métrica** que incorpore tanto nuestra reputación como la externa. La **reputación externa** se detalla en la capa centralizada, pues los datos se almacenan en la base de datos.

En los siguientes apartados se describen las implementaciones de las novedades mencionadas para obtener un robusto y eficiente sistema descentralizado. Acto seguido se muestran diagramas de secuencia para entender como se conecta cada sección en las funcionalidades de la aplicación.

IPFS[20]

Como hemos comentado anteriormente, para mejorar la eficiencia de nuestro proyecto a la hora de almacenar las denuncias decidimos introducir la tecnología IPFS. De una forma sencilla, podemos entender IPFS como un sistema distribuido para almacenar y acceder a archivos, sitios web, aplicaciones y datos. Entonces, en esta versión definitiva, los datos obtenidos directamente del formulario son almacenados en un archivo JSON que se almacena en IPFS. Este sistema lo que nos ofrece es un hash con el que sabemos donde se encuentra la denuncia dentro de esa red descentralizada. De esta manera, este hash es lo único que almacenaremos en el Smart Contract y, por tanto, en la red

de Rinkeby reduciendo considerablemente la cantidad de datos que almacenamos.

En nuestro proyecto, para poder acceder a esta red de IPFS, lo hacemos a través de Infura [127][128]. Esta plataforma nos ofrece una API HTTP RPC [129] ($/api/v0/*$) de IPFS que nos permite controlar el nodo y acceder al sistema de archivos. Para poder acceder a esta API, JavaScript nos ofrece la biblioteca *'ipfs-http-cliente'* [130]. De esta forma, con la biblioteca podemos crear una instancia del cliente API HTTP con la que poder usar los métodos ofrecidos y acceder a IPFS.

```
const client = ipfsHttpClient("https://ipfs.infura.io:5001/api/v0");
```

Una vez se envía un formulario y obtenemos los datos de la denuncia, creamos el archivo JSON con estos y lo añadimos a IPFS, como una cadena de caracteres, con la instancia del cliente y el método add, proporcionado por la API. Este método, es el que nos devuelve el hash que guardamos en el Smart Contract, a través del método *newComplaint*, y que necesario para luego acceder a la denuncia para obtener los datos a mostrar en las gráficas.

```
hash = await client.add(JSON.stringify(complaintJson));

//Relizar la denuncia
await newComplaint(Context.contract.methods.newComplaint(data.company, hash.path));
```

Por último, hacemos uso de esta tecnología a la hora de mostrar las gráficas. Para poder acceder a los datos de las denuncias obtenemos todos los hashes de las denuncia de nuestro contrato de Rinkeby. El método con el que accedemos a la denuncia en IPFS nos devuelve un vector de enteros que transformamos a formato JSON para recuperar todos los datos que nosotros mismos subimos anteriormente para poder ser manejados fácilmente.

```
for(const cid of complaintsHashes){
    for await (const chunk of client.cat(cid)) {
    // do something with buf
        let json = JSON.parse(uint8ArrayToString(chunk));
```

Tokens

El concepto de reputación es intangible. No existe forma empírica de medirlo. Se han desarrollado múltiples aproximaciones, basadas en el análisis de enormes cantidades de datos y comentarios, utilizando conceptos estadísticos. Sin embargo, en este proyecto se utiliza una nueva tecnología, actualmente en boca de todos, para tratar de representarla. Esta novedad son los tokens. En el estado del arte se muestran los principios que todo sistema de reputaciones debería seguir. Uno de ellos es que se generen expectativas de interacciones futuras y que exista una continua retroalimentación. Gracias al boom de los tokens, podemos aprovecharlos para fortalecer el interés sobre la aplicación.

Entre todos los que existen actualmente, destacan dos tipos[131]: fungibles y no fungibles. Los fungibles son análogos a las monedas, pues todos son exactamente igual. En cambio los no fungibles (más conocidos como NFT's) son únicos, representan un activo digital distinguible al resto. En el caso de la reputación, dos denuncias a diferentes empresas son igual de importantes, de la misma forma que una valoración de 7.5/10 es exactamente eso independientemente de la empresa. Por ello, desde nuestro punto de vista la reputación se asemeja más a al concepto de los tokens fungibles.

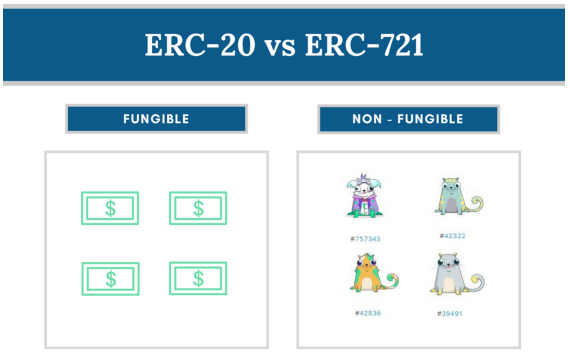


Figura 65: ERC-20 (fungible) vs ERC-721 (no fungible)
Tomado de [132]

Por otra parte, la reputación tiene una característica que choca frontalmente con el concepto de tokens: **no transferibilidad**. Carece de sentido que una empresa pueda comprar su reputación. De hecho, esto haría que nuestro sistema dejase de tener valor ¿Qué significa que una empresa tenga un 10/10? ¿Lo ha comprado o se lo ha ganado? En cambio, cualquier sistema descentralizado basado en tokens busca que se ejecuten las máximas transferencias de tokens para que el sistema mercantil coja fuerza, los tokens suban de valor y la DApp sea más conocida [133].

Por lo tanto, para representar el sistema de reputaciones, pretendimos emplear tokens no transferibles. Actualmente existen EIP's que se encuentran en estado de Draft, es decir, están en revisión esperando a ser modificadas. Todavía no son reconocidos por toda la comunidad de Ethereum y se desaconseja su uso. Estos son EIP-4671 y EIP-4974.En [134] puede consultar el estado de todas las propuestas actuales. No existe ningún ERC (estándar ya reconocido) que se ajuste a los requisitos de nuestra aplicación. Así pues, se han desarrollado dos soluciones posibles, ambas con ventajas y desventajas. La primera está basada en el estándar ERC20 y la segunda en el EIP4974, pues se implementó el contrato antes de que pasará a estado de Draft. Esto muestra lo cambiante que es el mundo Blockchain actualmente, ya que es posible que dentro de un mes dicho EIP pase a ser un estándar reconocido.

De todas formas, en las dos soluciones se sigue la misma estructura. Las denuncias se relacionan directamente con los tokens que recibe cada empresa. Esto significa que una denuncia equivale a un token. Por lo tanto, los tokens reputacionales son negativos. Cuántos más tenga una empresa, peor será su reputación. Para desarrollarlas se han utilizado materiales auxiliares, que se detallan en sus correspondientes secciones.

Las dos soluciones se han desarrollado en ficheros diferentes: *ReputationControlV2.sol* (ERC20) y *ReputationControlV3.sol* (ERC4974). Los contratos principales son prácticamente iguales, salvo las llamadas a los contratos de los tokens. Ambas opciones se han desplegado en la blockchain. Se han probado y funcionan correctamente. Las llamadas tienen los mismos nombres para que cambiar de una a otra en el Front sea muy sencillo, solo con modificar la dirección del contrato principal en el .env serviría. En la versión definitiva decidimos preservar el ERC20, pues su uso es seguro y está reconocido por toda la comunidad de Ethereum.

A continuación se describen ambas propuestas, y al final una sección con otras posibilidades que no siguen los requisitos iniciales y no se han implementado, pero que podrían ser útiles en el futuro de la aplicación.

1. Token ERC-20 [26]

Estándar por excelencia de los tokens fungibles. Este en realidad no es más que un contrato, ERC20, que implementa funciones prefijadas en la interfaz IERC20. Todos los contratos de los tokens y las interfaces necesarias que no hemos implementado nosotros se han descargado de la web OpenZeppelin [135]. Estas son las funciones de IERC20:

- *totalSupply()*: devuelve cantidad de tokens en el sistema.
- *balanceOf(address account)*: devuelve tokens de *account*.
- *transfer(address to, uint256 amount)*: transfiere *amount* tokens desde el llamante a la cuenta *to*
- *allowance(address owner, address spender)*: devuelve la cantidad de tokens que el *spender* puede gestionar de la cuenta de *owner*.
- *approve(address spender, uint256 amount)*: el llamante autoriza al *spender* para gestionar *amount* tokens suyos.
- *transferFrom(address from, address to, uint256 amount)*: transferir *amount* tokens desde *from* a *to*.

Al tratarse de tokens no transferibles, tan solo van a ser útiles las funciones de totalSupply y balanceOf. En la implementación del contrato ERC20 de OpenZeppelin no están especificadas funciones para crear y destruir tokens, aunque sí se incluyen dos funciones internas que cumplen dichos requisitos:

- *_mint(address account, uint256 amount)*: crea *amount* tokens nuevos y se transfieren a *account*
- *_burn(address account, uint256 amount)*: destruye *amount* tokens de la cuenta *account*

Todas estas funciones tienen sus propios require para controlar los posibles fallos. Al ser un ERC muy extendido, el código ha sido revisado múltiples veces y está libre de errores. Para poder llamar a las 2 funciones internas, se necesita otro contrato que herede de ERC20. Este nuevo contrato es el que se conecta con el principal para el reparto de los tokens, e implementa las 2 funciones (públicas) necesarias: *mint()* y *burn()*. Estas en realidad tan solo son llamadas a las internas. Además, para que otro contrato no pueda transferir tokens, se sobreescriben tanto *transfer()* como *transferFrom()*, añadiéndoles el modifier onlyOwner. Así, el contrato principal, que es el owner del contrato de los tokens, es el único que puede modificar los tokens que tienen las empresas. A continuación se muestra el código de este contrato, de nombre ReksTokens. Este es el nombre de los tokens.

```
1  // SPDX-License-Identifier: CC0
2  pragma solidity ^0.8.0;
3
4  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6  contract ReksTokens is ERC20{
7
8      address immutable owner;
9      modifier onlyOwner{
10         require(msg.sender == owner, "No permission");
11         _;
```

```
12      }
13
14      constructor(string memory _name, string memory _symbol) ERC20(_name, _symbol){
15          owner = msg.sender;
16      }
17
18      function mint(address to, uint256 amount) public onlyOwner{
19          _mint(to,amount);
20      }
21
22      function burn(address to, uint256 amount) public onlyOwner{
23          _burn(to,amount);
24      }
25
26      function transfer(address to, uint256 amount) public override onlyOwner
            returns (bool) {
27          address _owner = _msgSender();
28          _transfer(_owner, to, amount);
29          return true;
30      }
31
32      function transferFrom(
33          address from,
34          address to,
35          uint256 amount
36      ) public override onlyOwner returns (bool) {
37          address spender = _msgSender();
38          _spendAllowance(from, spender, amount);
39          _transfer(from, to, amount);
40          return true;
41      }
42  }
```

2. Token EIP-4974 [25][136]

Esta propuesta de estándar representa tokens fungibles no transferibles. Aún es una propuesta, pues todavía no está reconocida por la comunidad (en realidad es EIP, no ERC). Se conoce con el nombre de **Experience (EXP) Token Standard**, pues la idea es que sea análogo a los puntos de experiencia que se asignan en videojuegos, de tal forma que dicha experiencia no se puede canjear. Sin embargo, en nuestra aplicación le damos una connotación negativa, aunque el funcionamiento es esencialmente idéntico: crear y destruir tokens sin poder transferirlos. El contrato ERC4974 implementa las funciones descritas en la interfaz IERC4974:

- *setOperator(address operator)*: establece el operador a *operator*. Este es el único que puede operar con los tokens.
- *setParticipation(address participant, bool participation)*: añade o quite del sistema a *participant*.
- *transfer(address from, address to, uint256 amount)*: transfiere *amount* tokens de *from* a *to*. Si *from* es igual a *address(0)* entonces se crean los nuevos tokens que se transfieren a *to*. En caso de que *to* sea el *address(0)*, se destruyen los tokens de *from*.
- *totalSupply()*: devuelve cantidad de tokens en el sistema.
- *balanceOf(address participant)*: devuelve los tokens de *participant*.

El código del token se ha descargado del github del autor [137]. A diferencia del ERC20, todas las funciones están pensadas para la lógica de los tokens no transferibles. Por ello, no es necesario crear un nuevo contrato que herede de ERC4974, directamente el contrato principal se comunica con este.

3. Otras posibles soluciones

Ambas soluciones están basadas en el mismo concepto de reputación, la intransferibilidad. Además, están en consonancia con dos bases de la aplicación: los usuarios no pagan por nada y cada denuncia tiene el mismo peso. Si la elección de diseño de la aplicación hubiera sido que los usuarios deben costear todos los gatos, suena interesante que los tokens sean transferibles, estableciendo un ecosistema que favorezca al intercambio. Se emplearía el ERC20 en su totalidad. Algunas de las ideas que surgieron en este modelo fueron:

- Los tokens reputacionales pasan a tener connotación positiva. Esto significa que poseer muchos es beneficioso para la empresa.
- Denunciar supone un coste de x tokens. De hecho, esta cantidad puede aumentar con el nº de denuncias, para evitar más denuncias falsas.
- Se pueden apoyar denuncias aportando tokens. Cuantos mas tokens de apoyo reciba una denuncia, más tokens perderá la empresa. De partida, la empresa pierde una cantidad al ser denunciada.
- Si la empresa demuestra que el problema se ha solucionado, se le devuelven los tokens y se le entregan los apoyados, para incentivar la resolución de los conflictos.
- Los usuarios pueden comprar, vender y gastar los tokens, que tienen un precio inicial fijado.
- Las empresas no pueden transferir los tokens.

Contrato principal

Debido a las modificaciones con respecto a la versión inicial, el contrato central, *ReputationControl*, que controlaba las denuncias y la reputación, también ha cambiado. Ha tenido que adaptarse a la inclusión de los tokens e IPFS. Además incluye una nueva función que permite retirar tokens cuando se elimina una denuncia, por ejemplo si una empresa demuestra que ha solucionado el problema o que la denuncia era falsa. Los dos contratos principales, tanto el de los tokens ERC20 como el de los ERC4974, se encuentran en el Anexo, en la sección Versión Definitiva. Ambos son muy parecidos, tan solo se diferencian en las llamadas a las funciones de los contratos que guardan los tokens. Se recomienda encarecidamente leer uno de los dos para poder comprender mejor los siguientes párrafos.

De una empresa se almacena su dirección (ficticia, generada por el propio contrato, a no ser que tenga una cuenta); una lista de hashes que se corresponden con las denuncias que ha recibido y que no han sido resueltas; un mapping auxiliar para el borrado de las denuncias, que indica el índice de cada hash en la lista; y la posición que ocupa el nombre de la empresa en el array de empresas. Más adelante se entenderán mejor sus funciones. La dirección es necesaria en el manejo de los tokens, pues los estándar ERC funcionan a través de direcciones.

A parte de la información de cada empresa, *ReputationControl* mantiene una instancia del contrato encargado de los tokens, bien *ReksTokens* en el caso ERC20 o bien el propio ERC4974 en la segunda opción. Así consigue comunicarse con el sistema de tokens para que esté en concordancia con las denuncias. En ambas opciones, el operador del contrato de los tokens es el propio *ReputationControl*, lo que significa que solo él puede ejecutar manejarlos. Además, también guarda una variable que almacena la cantidad de ethers donados a la aplicación. Como en Solidity no existen floats, se utiliza un string.

En cuanto a las funciones, se han añadido numerosas y el resto ha cambiado su implementación con respecto a la versión inicial. Se explican paso a paso todas los modificaciones:

- Al crear nuevas empresas, la reputación inicial es 0 y se han de registrar en el contrato de los tokens.
- Se añade una función, *retireCompany(string memory company)* que permite borrar una empresa del sistema. Tan solo la puede ejecutar el propietario del contrato y el nombre pasado como argumento debe pertenecer al sistema. La operación se basa en quemar todos los tokens que tenía, borrar toda su info y eliminar su nombre del array.

  Para eliminar el nombre de la lista, se intercambia con la última posición y se hace un pop. Es en este punto cuando se utiliza la variable *pos*, almacenada en el struct de la empresa. Así se reduce el nº de accesos a storage, pues la otra opción sería recorrer el array, lo que supone un coste de $O(n)$ accesos. Con este tipo de implementaciones conseguimos que el consumo de gas de cada transacción sea menor, es decir, ahorramos dinero.
- Otra buena praxis para reducir los accesos a storage es emplear variables locales, que se guardan en memory. Si un dato de storage se accede más de 1 vez en una función, utilizamos la redundancia de variables para ahorrar gas.
- Al añadir una denuncia, se apunta el índice que ocupa esta en la lista de denuncias de una empresa. Se utiliza para eliminarla. Además, la empresa recibe un token.
- Se añade una función, *retireComplaint(string memory company, string memory hash)*, que permite borrar una denuncia. Al igual que sucede con el nombre de la empresa, para borrar el hash se intercambia con la última posición y se hace el pop. Los índices necesarios para el cambio se almacenan en el mapping de hashes a índices de la propia empresa. Por lo tanto, también hay que actualizarlo.

  Un problema con los mappings es que a cualquier hash que no tenga valor, le da por defecto el 0. El índice 0 representa que ese hash no existe, y las posiciones empiezan en 1. Es por esto que se almacena el índice + 1 y a la hora de acceder a la lista de hashes se tiene en cuenta.

  De nuevo al igual que en la función *retireCompany*, se utiliza el mismo concepto de guardar las posiciones para evitar recorrer el array de hashes, que supone un coste $O(n)$ de accesos a storage, por los 6 que contiene la función.
- Se añaden funciones para reflejar las donaciones en la variable que las almacena y para devolverla.

Despliegue del contrato

Al igual que en la versión inicial, tanto la compilación como el despliegue se han realizado desde local. De han reutilizado los dos ficheros *compile.js* y *deploy.js*. Sin embargo, para poder compilar el fichero *ReputationControlV2.sol*, que contiene la lógica del sistema de reputación con tokens ERC20, se han introducido dos cambios en la compilación.

El primero de ellos es resolver las dependencias del import. Como ya se ha mencionado antes, los contratos auxiliares como ERC20 se han descargado en la carpeta *node_ modules*. Sin embargo, la extensión de Solidity utilizada no es capaz de seguir las rutas. Es necesario indicarlas en el fichero *settings.json*. Además hay que modificar otras configuraciones del propio compilador de Solidity. En [138] se han encontrado las soluciones a este problema. El segundo es añadir unas líneas de código que permiten encontrar las rutas de los ficheros importados. También se modifica la llamada *compile()* para que tenga en cuenta dichas rutas. A continuación se muestran las modificaciones con respecto al primer *compile.js*

```
function findImports(relativePath) {
  //my imported sources are stored under the node_modules folder!
  const absolutePath = path.resolve(__dirname, 'node_modules', relativePath);
  const source = fs.readFileSync(absolutePath, 'utf8');
  return { contents: source };
}

//Compilamos ambos contratos
const output = JSON.parse(solc.compile(JSON.stringify(input), { import:
    findImports })).contracts['ReputationControlV2.sol'];
```

El deploy se modifica de acuerdo con los nuevos nombres de los ficheros. Además, ejecuta la función newCompanies() para añadir las 10 empresas iniciales que se toman como muestra para el prototipo. Como ya se ha mencionado anteriormente, los dos contratos principales, junto con sus tokens, se han lanzado a la red de Rinkeby y ambos han sido testeados.

Métrica

Los **Reks** representan la reputación interna de nuestra aplicación, a través de las denuncias que se emiten. Sin embargo, como ya se ha mencionado antes, un sistema de reputación también debe ser robusto y representativo al comienzo de su vida. Es decir, en las primeras etapas de la web, aunque no hayan muchas denuncias, el valor reputacional debe tratar de reflejar lo que sucede en realidad. Por lo tanto, las reputaciones que aparecen en la interfaz dependen de 3 componentes: **tokens** y dos **reputaciones externas** a nuestra DApp, procedentes de Glassdoor[139] e Indeed[140]. Para calcular el valor total se aplica una serie de fórmulas. Primero, los tokens se introducen en una ecuación, obteniendo así el valor de **reputación interna**, R_{int}.

$$R_{int} = 5 * (1 - \frac{n_i}{N}) + 5 * e^{-n_i/K}$$

Donde n_i es es el nº de Reks de la compañía y N el nº total de Reks. Observar que $R_{int} \in (0, 10]$, como se desea. Cada sumando expresa un concepto diferente:

- El primero compara cuantas denuncias ha recibido una empresa con respecto al resto. Si la compañía ha recibido pocas denuncias, no se sabe si en realidad son pocas o es que casi nadie

ha denunciado. Sin embargo, este sumando por si solo no es suficiente. Puede darse el problema de que todas las empresas sean "malasz reciban muchas denuncias, en la misma proporción. En ese caso, aun habiendo recibido múltiples Reks, todas tendrían una elevada reputación porque los porcentajes serían bajos (similares todos pero bajos).

- El segundo decrece si el nº de denuncias que recibe la empresa aumenta. Este sumando trata de compensar el efecto descrito en el primer sumando. La constante K expresa de alguna forma el nº de denuncias que consideramos elevado. En principio está en $K = 20$, aunque puede ser modificada en cualquier momento.

Una vez calculada R_{int}, se aplica una media ponderada [141] con las reputaciones externas para obtener la reputación total R_t:

$$R_t = \alpha_0 * R_{int} + \alpha_1 * (r_{gls} + r_{ind})$$

Los pesos α_i van cambiando a lo largo del tiempo, en función del volumen de denuncias que gestione la aplicación. En la fase inicial solo se tiene en cuenta las externas, pues la reputación interna tendrá un valor muy elevado. Según vaya creciendo la web, R_{int} irá cobrando importancia, es decir, α_0 aumentará. Los valores estipulados son:

- $0 \leq N < empresas : \alpha_0 = 0$ y $\alpha_1 = 0{,}5$
- $empresas \leq N < 3 * (empresas) : \alpha_0 = 0{,}1$ y $\alpha_1 = 0{,}45$
- $3 * (empresas) \leq N < 5 * (empresas) : \alpha_0 = 0{,}2$ y $\alpha_1 = 0{,}4$
- $5 * (empresas) \leq N : \alpha_0 = 0{,}35$ y $\alpha_1 = 0{,}325$

7.2.2. Sistema Centralizado

En la versión definitiva, se partió de la versión inicial y se añadieron mejoras, funcionalidades y nuevas rutas de forma que el Front-End pudiera hacer peticiones para obtener distintos tipos de datos requeridos para las nuevas necesidades de la aplicación.

Almacenamiento de Sesiones

En la versión original se guardaban las sesiones de los usuarios en el almacenamiento local del servidor [111]. Esto significaba que si se producía un fallo en el servidor y este se detenía, las sesiones se perderían y tendrían que volver a iniciar sesión incluso si estaba iniciada en momentos previos a la caída. Para solucionar esto, se incorporó *connect-mongo* [142] al proyecto, el cuál se encarga de almacenar y gestionar las sesiones en una base de datos MongoDB.

Se conectó a nuestra base de datos de Mongo Atlas, creando automáticamente una colección llamada *sessions* con el fin de almacenar ahí las sesiones hasta que el usuario cierre sesión o se alcance su fecha de expiración, siendo en nuestro caso una hora tras la creación de la sesión. A su vez, se establecen los valores de *saveUninitialized* y *resave* [111] para que solo se almacenen aquellas sesiones resultantes de haber realizado el login y no se reinicien tras cada petición.

En el siguiente fragmento de código se muestra la configuración final de las sesiones:

```
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false,
  store: MongoStore.create({ mongoUrl: mongoose.connection._connectionString }),
  cookie: { expires: expiryDate,  httpOnly: true }
}));
```

Datos para gráficas

Para poder mostrar gráficas sobre distintas métricas recogidas dentro de la aplicación se decidió almacenar información sobre distintos campos del formulario de denuncia en la base de datos.

En primer lugar, se creó una colección llamada *chartData* y se definió el esquema y el modelo de Mongoose con el que serían almacenadas y obtenidas las gráficas:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const ChartDataSchema = new Schema({
    category : { type : String},
    value : { type : Number }
});

ChartDataSchema.set('autoIndex', false);

const ChartSchema = new Schema({
  chart_name : { type : String, required : true, unique : true},
  data : [ChartDataSchema]
}, { collection : 'chartData' });

const Chart = new mongoose.model('chartData', ChartSchema);
module.exports = Chart;
```

Cada documento de gráfica posee un nombre, definido por *chart_name* y un array de objetos denominado *data*. Los objetos que componen el array tienen dos campos:

- *category*: Contenido del campo del formulario de denuncia pertinente a la temática de la gráfica. Por ejemplo, femenino sería una categoría perteneciente a una gráfica de género.
- *value*: Número de denuncias hay con esa categoría.

Una vez definidos el esquemas y el modelo, se decidió el número de gráficas deseadas y la temática de estas ya que deben ser inicializadas para que puedan aparecer todas las posibles categorías en la leyenda. La decisión resultante fue que se crearían cuatro gráficas que mostraran datos sobre género, edad, tipo de denuncia y consentimiento para leer la historia del denunciante.

Para inicializar las gráficas, se creó una función llamada *initChartData()* que se ejecuta exclusivamente en caso de que la colección chartData esté vacía. Esta función crea cuatro documentos de gráfica (graficaTipo, graficaEdad, graficaGenero y graficaRelacion) con sus arrays *data* poblados con todas las categorías posibles y con valor 0.

De forma que se pueda interactuar con estos datos se creó una nueva ruta llamada */charts*. La ruta posee dos endpoints, uno para GET y otro para POST. El endpoint para peticiones GET, */charts/:chartName*, recibe el nombre de una gráfica a modo de variable de ruta, la cuál se busca en la base de datos y en caso de encontrarla exitosamente devuelve el campo *data* con las respectivas categorías y valores.
Por otro lado, el endpoint para peticiones de tipo POST, */charts* recibe del formulario los valores de los campos tipo, edad, género y relación actual con la empresa. Posteriormente se llama a función updateChart(chartName, category) donde para cada gráfica se pasa el campo correspondiente y se busca una coincidencia con alguna categoría. Una vez se encuentra la categoría se aumenta en uno campo valor del elemento coincidente.

Reputación Externa

Debido a que la reputación inicialmente provenía en su totalidad de las denuncias, en caso de no haber ninguna, la reputación de las empresas se vería en su máximo valor. Por tanto, se decidió almacenar valoraciones de las distintas empresas provenientes de Glassdoor e Indeed, dos portales de búsqueda de empleo donde los usuarios pueden escribir reseñas sobre su experiencia en la empresa. De esta forma, además de la reputación interna de la aplicación web se tiene en cuenta lo que denominamos *reputación externa*: la reputación generada a partir de usuarios ajenos a nuestra plataforma.

Al igual que en con los usuarios y las gráficas, la primera acción a realizar fue crear la colección *companyMetrics*. Mas adelante se crearon el esquema de los datos y el modelo que lo instancia:

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  const MetricsSchema = new Schema({
5    company_name : { type : String, required : true, unique : true},
6    rating : { type : Array, required : true, unique : false}
7
8  }, { collection : 'companyMetrics' });
9
10 const Metrics = new mongoose.model('companyMetrics', MetricsSchema);
11 module.exports = Metrics;
```

En la colección se almacenan la reputación externa mediante los dos campos que se pueden observar. El campo *company_ name* contiene el nombre de la empresa y rating posee el array con las dos valoraciones provenientes de las webs.

La extracción de la información de las plataformas se realiza mediante un método de *scraping* [89], el cuál consta de la obtención de un recurso web y el contenido deseado de este. Para ello, se crea un script de Python que emplea *Beautiful Soup* [143], una librería que permite extraer datos de ficheros XML y HTML. Además se utiliza *MongoClient* [144] para poder realizar la conexión con la base de datos.

El script en primer lugar se conecta a nuestra base de datos y recorre un bucle donde, en cada iteración envía una petición a la página de opiniones de Glassdoor de una empresa y otra a reviews de Indeed. Posteriormente, el código HTML que devuelven ambas peticiones es procesado por Beautiful Soup, al que se le indica el nombre de los componentes exactos que debe buscar en cada caso.

Finalmente, ambos contenidos obtenidos de los componentes se añaden a un array y este se envía a la base de datos junto con el nombre de la empresa, actualizando las valoraciones existentes.

Para que el servidor pueda ejecutar este script, se utiliza *python-shell* [45], un módulo que crea un proceso hijo donde se ejecuta una consola de Python. No obstante la ejecución, es una operación pesada, ya que se realizan dos peticiones por cada empresa y se debe buscar en cada una los elementos determinados a través de todo el código HTML. Por tanto, se utiliza en conjunto *node-cron* [44], un programador de tareas para Node.js, configurado para que se ejecute el script todos los días a las doce de la noche de forma que no se realice de forma intensiva pero manteniendo la reputación externa lo más actualizada posible.

Por último, la reputación externa de la empresa extraída mediante el proceso descrito puede ser obtenida a través de la ruta */getMetrics/*, mediante el endpoint para GET */getMetrics/:companyName*. Este endpoint recibe el nombre de una empresa a modo de variable de ruta y devuelve el campo rating con la valoraciones.

Documentación de los endpoints

Tras añadir funcionalidades al servidor, el número de rutas y por consiguiente, endpoints aumentó considerablemente. Para poder documentar y registrar las funciones que desempeña cada uno se decidió incorporar Swagger.

Se utilizó el módulo *swagger-jsdoc* [145], que permite detectar anotaciones de Swagger para generar una especificación, que define los ficheros necesarios para describir una API. Estos ficheros son luego empleados por *swagger-ui-express* [146] para generar la documentación y servirla en la ruta del servidor */swagger*.

Las anotaciones [147] se realizan a modo de comentario, encima de la creación del endpoint. Mediante estas se puede detallar la ruta, tipo de endpoint, si se necesita algún tipo de variable de ruta o incluso ejemplos de resultado. Además, permite crear tipos de datos [148] para ser referenciados en el body de la petición o como resultado esperado.

A continuación se muestra un ejemplo de anotación:

```
/**
* @openapi
* components:
*   schemas:
*     User:
*       type: string
*
*
* @openapi
* /getuser:
*   get:
*     tags:
*       - Login
*     summary: Obtiene el nombre de usuario.
*     description: Devuelve el nombre de usuario para enviarlo al Front.
*     responses:
*       200:
*         description: Nombre de usuario recibido con exito.
*         content:
```

```
*            application/json:
*              schema:
*                type: string
*                $ref: '#/components/schemas/User'
*              example: ZcNJA69bAkceP31
*/
```

Al acceder a la ruta del servidor donde se sirve el documento, se puede observar el siguiente resultado:

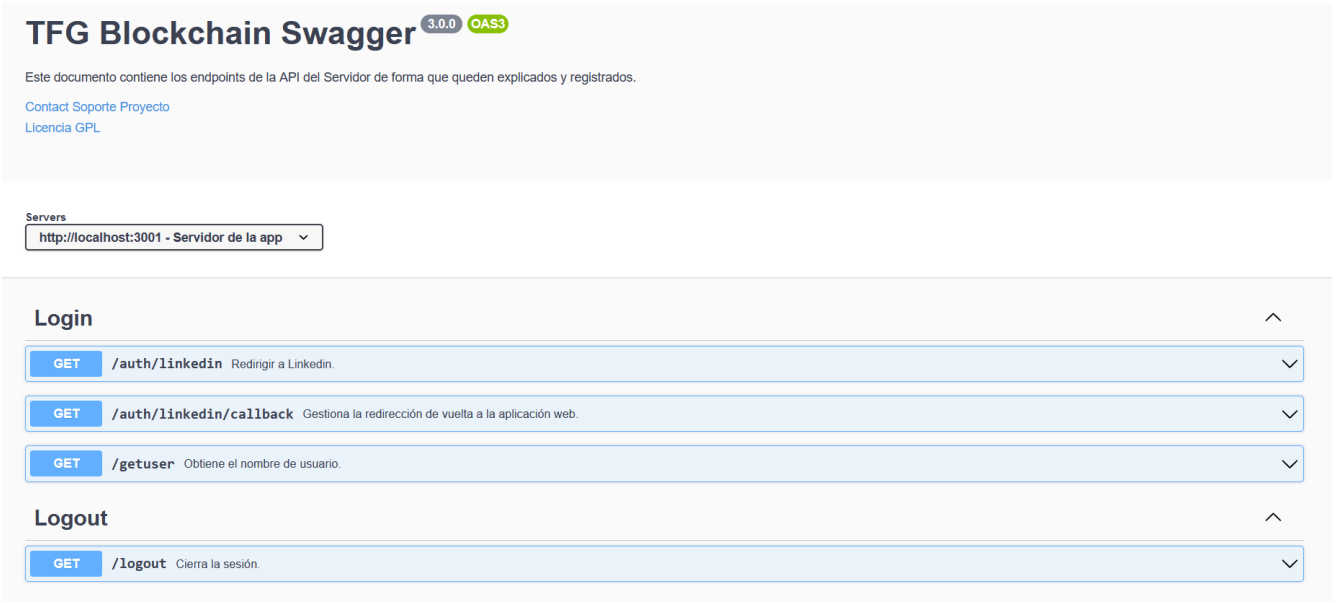


Figura 66: Documento Swagger
Fuente propia

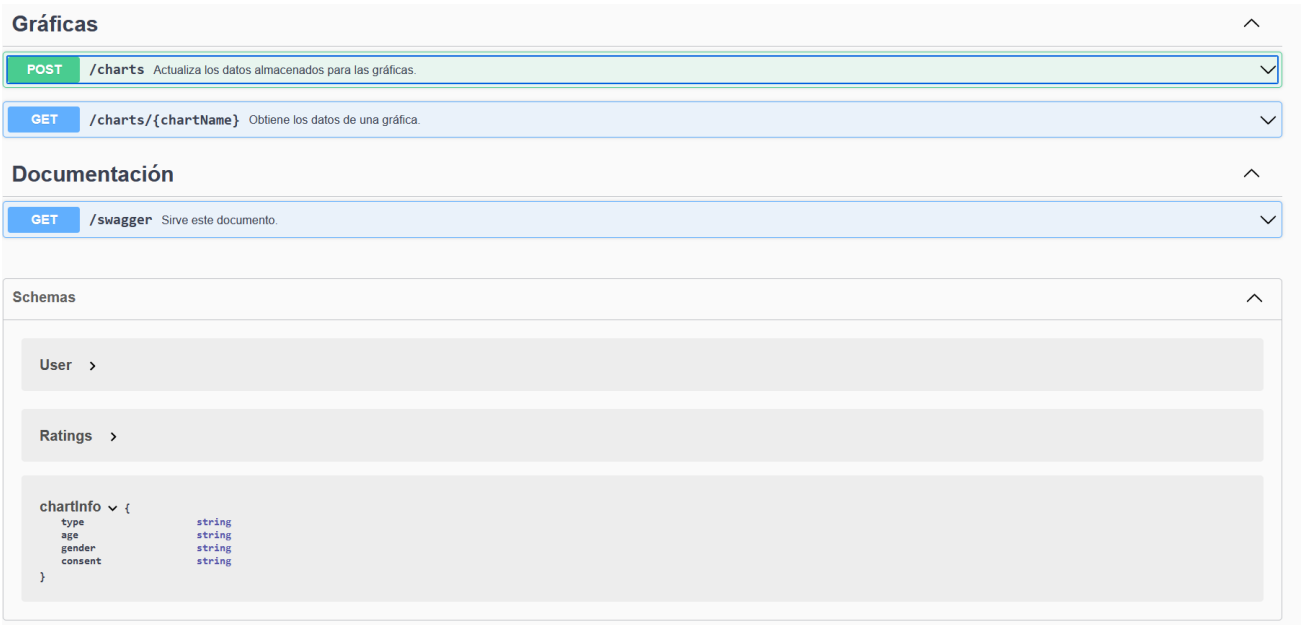


Figura 67: Documento Swagger: Schemas
Fuente propia

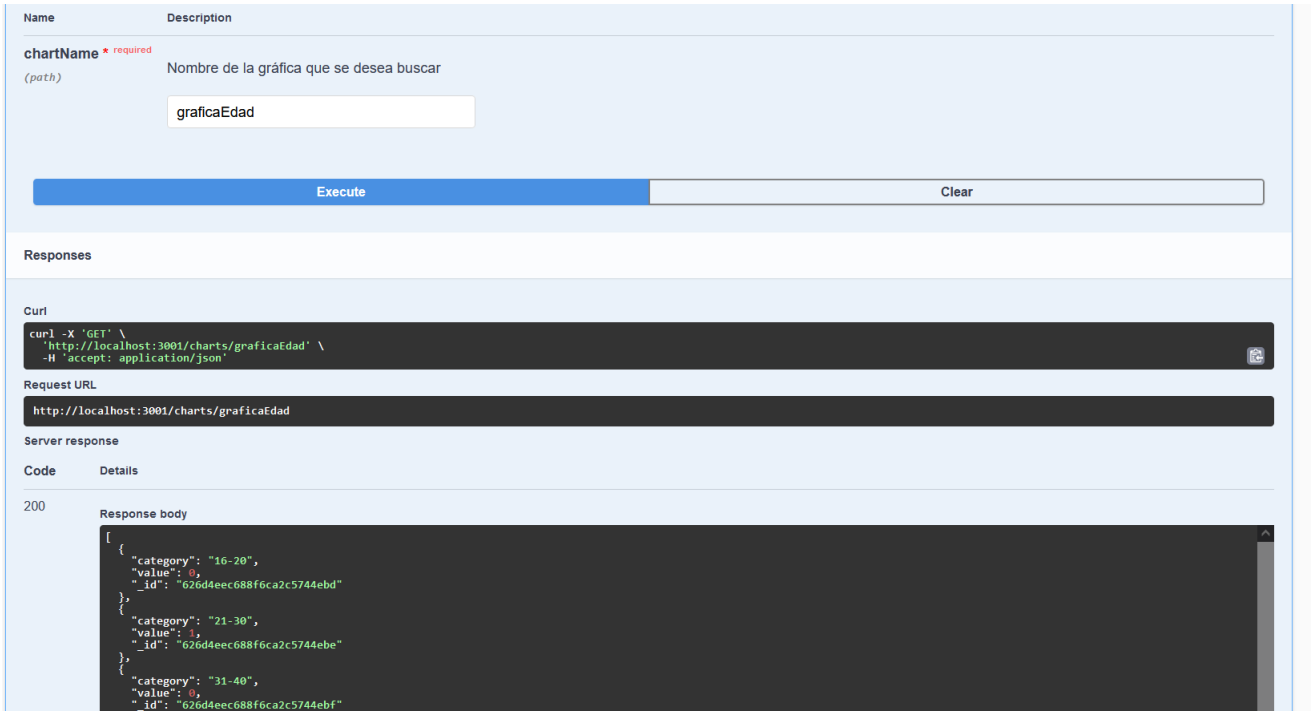


Figura 68: Documento Swagger: Testing
Fuente propia

De esta forma, al usar Swagger se puede acceder a la documentación con facilidad y además de poder probar el correcto funcionamiento de cada endpoint sin tener que exponer el código.

Dockerización

Con el objetivo de aumentar la portabilidad y escalabilidad del proyecto se decidió incluir Docker al repertorio de tecnologías empleadas.

Para poder crear la imagen del servidor, se debe crear un *Dockerfile* [149]. Este es un fichero que contiene todos los comandos necesarios para construir la imagen:

```
1 FROM node:16-alpine3.14
2 RUN mkdir /server
3 WORKDIR /server
4 COPY package*.json ./
5 RUN npm install
6 COPY . .
7 EXPOSE 3001
8 CMD ["npm", "start"]
```

Se parte de una imagen de Node.js basada en *Alpine* [150], una distribución de Linux de forma que sea lo más ligera posible ya que esta ocupa aproximadamente 5MB. Gracias a esto, se podrían utilizar varias imágenes sin consumir grandes cantidades de almacenamiento, construyendo la arquitectura escalable mencionada con anterioridad. En adición, se indica que se cree un nuevo directorio llamado *server* y lo establezca como directorio de trabajo. En este directorio, se copia el package.json y package-lock.json y se instalan las dependencias mediante el primero. Posteriormente se copia el código del servidor. Por último se indica que escuche en el puerto 3001 y se inicie el servidor. El cliente también posee un Dockerfile que sigue instrucciones muy similares para poder crear una imagen.

Para crear una imagen del servidor y ejecutarla como contenedor se necesitaría realizar dos comandos [151]:

- *docker build -t <nombreImagen> .*
- *docker run -p 3001:3001 <nombreImagen>*

El primer comando construye la imagen a partir del Dockerfile indicado en la ruta, que en nuestro caso es '.' y le asigna el tag; mientras que el segundo ejecuta el contenedor y expone su puerto 3001 sobre el puerto 3001 de la máquina.

Junto con los Dockerfiles, se creó un archivo de Compose [152], el cuál permite crear varias imágenes a la vez y definir una red de forma que puedan comunicarse entre sí a través de esta.

En el siguiente fichero se muestra cómo, por ejemplo, se podría crear el servidor y cliente a la vez:

```
1 version: "3.7"
2
3 services:
4
5   server:
6     build: ./server
7     image: server:latest
8     ports:
```

```
9         - 3001:3001
10
11    cliente:
12      build: ./cliente
13      image: cliente:latest
14      ports:
15        - 3000:3000
```

Dentro de services se indican las imágenes que se desean crear indicando [152]:

- *build*: Ruta del Dockerfile con las instrucciones sobre cómo montar la imagen.
- *image*: Nombre de la imagen y su correspondiente versión.
- *ports*: Puertos del tipo *HOST:CONTENEDOR*, indicando que el puerto CONTENEDOR se comunicará con el exterior mediante el puerto HOST de la máquina que lo ejecuta.

Solo sería necesario realizar el comando *docker-compose up* [66] para crear la red interna, las imágenes y ejecutarlas a modo de contenedores. Si se deseara crear varios servidores para escalar la aplicación mediante Docker Compose solo haría falta declarar en el fichero más imágenes del servidor escuchando en puertos distintos.

Figura 69: Multicontenedor en ejecución
Fuente propia

En el siguiente apartado se muestra la adaptación del sistema de forma que opere con múltiples servidores, tal y como se describió en el punto 5.1.2.

Balanceo de carga y tolerancia frente a fallos

La dockerización del proyecto, tal y como se ha mencionado previamente, da cabida a una fácil escalada del mismo mediante la creación de un sistema con múltiples servidores.

Para empezar, es necesario la creación de un *balanceador de carga* [153], cuya función es la de actuar como punto de contacto con el cliente y distribuir las peticiones entre distintos servidores, cuyo conjunto se denomina *server pool*. La herramienta utilizada en nuestro proyecto es *Nginx* ya que posee una imagen de Docker lista para su uso en el *Docker Hub* [154], un repositorio de imágenes.

Se escribió un Dockerfile, que indica que se cree una imagen a partir de la imagen de Nginx, borre el fichero default.conf y finalmente copie el contenido del fichero de configuración creado sobre un nuevo default.conf:

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

El fichero nginx.conf, contiene las instrucciones de configuración de nuestro balanceador de carga especificadas en el módulo [155] *ngx_http_upstream_module*. Mediante *upstream* se establece un cluster con los servidores definidos a modo de contenedores de Docker junto a su puerto interno, a los que el balanceador puede delegar el tráfico. Dentro del cluster se establece la estrategia de repartición de tráfico a utilizar, siendo *least_conn* la elegida de forma que el servidor con menos peticiones sea el que se ocupe de atender la siguiente. Posteriormente, se levanta un servidor que escucha en el puerto 3001 y que redirige el contenido al cluster. También se configura para que en caso de producirse un error o caída y no se pueda realizar una conexión con el servidor asignado en 2 segundos, se dirija el tráfico a otro servidor.

A continuación se muestra el contenido de nginx.conf:

```
upstream servers {
  least_conn;
  server server_1:3001;
  server server_2:3001;
  server server_3:3001;
}

server {
  listen 3001;
  location / {
    proxy_pass http://servers;
    proxy_set_header Host $host:$server_port;
    proxy_next_upstream error timeout invalid_header http_500;
    proxy_connect_timeout 2;
  }
}
```

Una vez creados el Dockerfile y su respectivo archivo de configuración de Nginx, se modificó el archivo de Compose definido en el apartado anterior para que pudiera crear el balanceador de carga y las distintas imágenes del servidor:

```
version: "3.7"

services:

  #aplicacion
  cliente:
    build: ./cliente
    image: cliente:latest
    ports:
      - 3000:3000

  #server pool
  server_1:
    build: ./server
    image: server:latest
    ports:
      - 3002:3001
```

```
18      restart: on-failure:3
19
20    server_2:
21      build: ./server
22      image: server:latest
23      ports:
24        - 3003:3001
25      restart: on-failure:3
26
27    server_3:
28      build: ./server
29      image: server:latest
30      ports:
31        - 3004:3001
32      restart: on-failure:3
33
34    #load balancer
35    nginx:
36      build: ./load_balancer
37      image: nginx:latest
38      ports:
39        - 3001:3001
40      depends_on:
41        - server_1
42        - server_2
43        - server_3
44      restart: on-failure:3
```

Como se puede observar, se crean tres contenedores que parten de la imagen del servidor. Cada contenedor se expone sobre un puerto distinto del host conectado al puerto interno 3001, donde la aplicación de Node.js está escuchando. El contenedor del balanceador de carga se expone sobre el puerto 3001 de forma que la aplicación pueda funcionar independientemente de su uso con la implementación de un solo servidor o multiservidor. En adición, mediante *depends_ on* [152] se instruye al balanceador para que antes de construirse espere a que los contenedores que componen el server pool estén listos ya que es necesario que estén definidos a la hora de crear el cluster.

Finalmente, se establece el valor de *restart* [152] en el balanceador de carga y servidores a "on-failure:3" de forma que en caso de fallo o caída se realicen tres intentos de volver a iniciar el servicio. Si un servidor cae, el resto de servidores se ocuparán de atender las peticiones que este recibiría. Además, al almacenarse las sesiones en una colección de la base de datos, cualquier servidor puede ocuparse de gestionar el tráfico proveniente de cualquier usuario. Esto resulta en la creación de un sistema con capacidad de recuperación, donde las caídas y el volumen de tráfico causan un menor impacto; garantizando la mayor disponibilidad y tolerancia a fallos posible.

7.2.3. Front

A la hora de llevar a cabo la transición entre la versión anterior y una nueva, necesitábamos una plantilla donde apoyarnos para conseguir una estética agradable. Para ello, hicimos research acerca de qué plantillas gratuitas estaban disponibles en Material UI, y elegimos Berry [59], aquella en la que creímos que iba a ser más fácil amoldar el trabajo que ya habíamos realizado durante la primera versión.

El primer paso tras descargar la plantilla fue la fase de diseño, en la que se realizó una reunión entre las dos personas encargadas del Front para fijar cómo iban a visualizarse todas las pantallas de la plataforma, basándose no solo en las que ya teníamos sino también en el diseño de la propia plantilla. El resultado del diseño fue el siguiente:

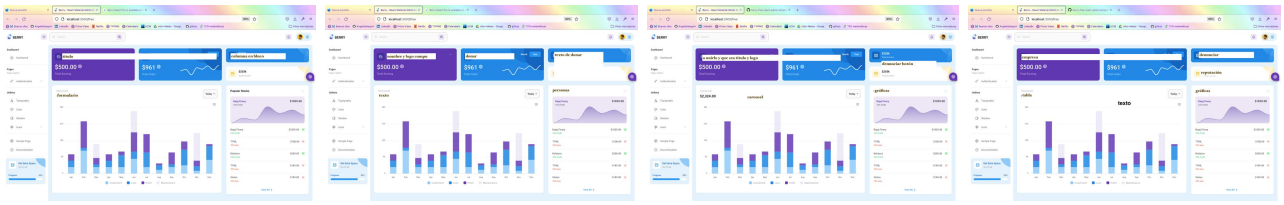


Figura 70: Boceto final de la página principal
Fuente propia

A continuación, comenzamos a realizar las modificaciones necesarias en la plantilla. Primero aquellas relacionadas con el índice, el formato, o las imágenes para ir a la pantalla principal. A posteriori, nos centramos en el diseño principal de nuestra plataforma. A continuación, detallaremos cómo ha sido la transición de cada una de las pantallas a la plantilla. La plantilla solo constaba de una pantalla, por lo que nuestra labor fue hacer copias de esta pantalla y darlas forma de manera individual.

- **Pantalla Principal:**

  El desarrollo del nuevo Front para la plataforma se comenzó con esta pantalla. Inicialmente se configuró el índice con las opciones necesarias y se establecieron las rutas correctas para redirigirnos a las demás páginas que se implementarían más tarde. Tras esto, se realizó un rediseño y reajuste de los componentes presentes en la plantilla. En primer lugar, se mantuvo el header de la plantilla que sufrió una serie de modificaciones:

  - Se eliminó el componente de búsqueda, pues no era necesario en nuestra plataforma
  - La sección derecha correspondiente al perfil se modificó por completo. El componente en sí mantuvo su diseño cambiando la foto de perfil por el logo de LinkedIn sin embargo, la funcionalidad ahora consiste en permitir la autenticación de la cuenta de LinkedIn para poder interponer una denuncia. Al pinchar el botón aparece un recuadro emergente que nos da la opción de hacer *Login* o *Logout* dependiendo de si hemos accedido ya o no. En caso de clickar en Login nos redirige a la página de acceso a LinkedIn.
  - El botón que visualiza o esconde el menú principal se mantuvo igual y solo se cambió el logo. El menú en sí, como se ha destacado antes, se cambió para contener las opciones de navegación adecuadas a nuestro proyecto.

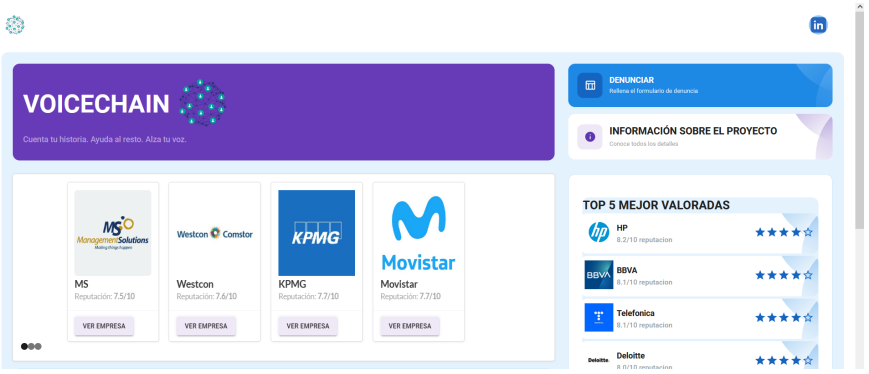
  Por otra parte, las componentes tarjetas que aparecían en la plantilla se mantuvieron sufriendo pequeños cambios en el diseño y en la funcionalidad. Como una de las partes principales de la pantalla de inicio se diseñó un Carousel automático que contiene una serie de tarjetas con la información de las empresas recogidas por la plataforma. Las tarjetas se diseñaron con componentes de la librería Semantic UI [156] y se agruparon convenientemente para ser mostradas en el Carousel. Otra de los componentes principales de la página de inicio consiste

es el componente **DataGraphics** que se trata de un contenedor de gráficas de diferentes tipos que muestran las estadísticas referentes a las denuncias realizadas. El tamaño, el número de gráficas, los colores y la información que se representan pueden ser modificados fácilmente mediante las props que se pasan al componente.

Finalmente, el último componente principal de esta pantalla es el ranking de empresas. Se trata de un componente que muestra el Top 5 de empresas mejor y peor valoradas según la reputación. Cada ranking contiene su título, así como 5 mini tarjetas con la información de cada empresa (logo, nombre y reputación) además de una escala de estrellas para representar visualmente la reputación. El número de estrellas totales son 5 y aparecen vacías, semirellenas o totalmente coloreadas gracias a una función que según el intervalo en el que se encuentre el valor de la reputación de cada empresa renderiza las estrellas de una forma u otra.

Para el diseño de esta pantalla se tuvieron en cuenta las **Leyes UX**, que establecen una serie de pautas de diseño web para mejorar la experiencia del usuario. Algunos ejemplos aplicados en el diseño de esta pantalla son los siguientes:

- **Principio de consistencia interna**: en los botones de donar y denunciar, pues son dos funcionalidades completamente diferentes así que se incluyen en botones de colores distintos.
- **Ley de Hick**: esta dice que el tiempo que tarda el usuario en tomar una decisión aumenta a medida que se incrementa el número de opciones. Por lo tanto, se decidió que se incluirían pocos caminos alternativos en esta pantalla: el menú tiene tan solo 3 pestañas y las funcionalidades principales que aparecen aquí son dos representadas en dos tarjetas al principio: denunciar y donar.
- **Ley de Jacob**: es la relativa a que los usuarios pasan la mayor parte de su tiempo en otros sitios web. Esto significa que prefieren sitios similares a los que frecuentan y ya conocen. Por eso, el equipo se decantó por no "pasarse"de originales y diseñar una página web visualmente agradable y atractiva pero sin incluir componentes demasiado raros que puedan llegar a cansar al usuario.
- **ley de Von Restorff**: el botón denunciar tiene un tamaño considerable y aparece en un color llamativo, el azul porque representa la funcionalidad principal de la plataforma.
- **Principio de cierre**: el componente de gráficas aparece entrecortado al iniciarse la página, lo que permite al usuario imaginar la existencia de más gráficas en la parte inferior.
- **Principio de consistencia externa**: al representar el grado de reputación mediante una escala de estrellas puesto que es un recurso muy utilizado y conocido por usuarios en muchísimas plataformas.



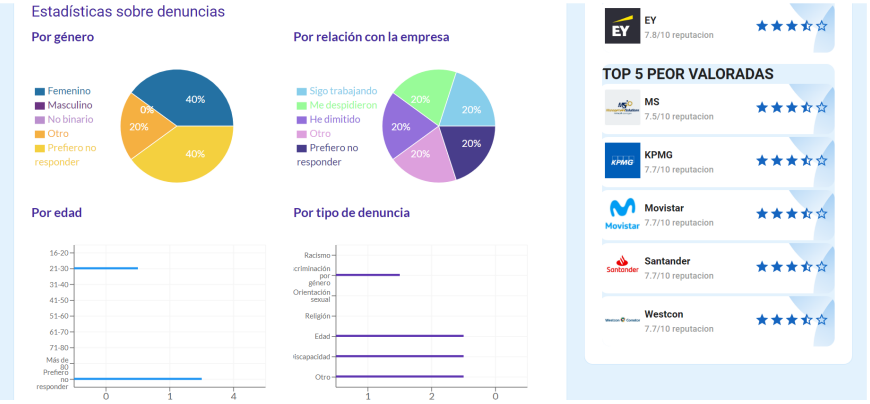


Figura 71: Diseño final de la página principal
Fuente propia

- **Pantalla para visualizar una empresa:**

Recordamos que una de las funcionalidades principales de esta pantalla es la denuncia a la empresa en cuestión. Queremos que este proceso se realice de la manera más rápida posible, en el menos número de clicks, y sea fácil de visualizar, por lo que optamos por un botón en la parte superior con un texto de tamaño suficiente y color llamativo, empleando la **ley de Von Restorff**, tratando de reforzar una acción por encima de las demás: llamar la atención del denunciante hacia la funcionalidad principal de la plataforma que es denunciar.

Asimismo, introducimos una nueva tarjeta en esta ventana que nos permite conocer la reputación que tiene esta empresa, basada en las denuncias realizadas y la métrica empleada. Dicha reputación aparece en un tamaño menos al botón de denunciar, tratando de resaltar el anterior.

Los cambios más significativos en esta pantalla se concentran en la tabla de denuncias. La librería empleada en la versión anterior tenía un diseño que no casaba con el de la nueva plantilla. Por ello, buscamos nuevas maneras en las que proveer a esta pantalla de un lugar donde poder ver las denuncias. EL resultado final fue una tabla formada por componentes de Material UI en la que, para poder identificar más fácil cada tipo de denuncia, aparece un avatar con la inicial del tipo correspondiente. Perdíamos funcionalidad al no tener, por ejemplo, capacidad para filtrar las denuncias como ocurría en la versión anterior, pero consideramos que era algo que no tenía mucha importancia pues no aportaba demasiado valor. En dicha tabla aparece el botón correspondiente para leer la experiencia. Si el usuario pulsa el botón en caso de querer leerla, aparece el texto correspondiente en un diálogo y el resto de la pantalla se oscurece.

Otro aporte significativo que proporciona funcionalidad y limpieza fue la introducción de gráficas en formato de pestañas. De esta manera, en lugar de visualizar todas las gráficas creadas con la librería Recharts [157] en la anterior versión a la vez, lo cual ocupaba una cantidad ingente de espacio, el usuario puede seleccionar la gráfica concreta que quiere ver dentro de una variedad de pestañas con título que indica el tema principal que abarca dicha gráfica. El diseño de la gráfica coincide con el generado para la versión anterior.

Figura 72: Diseño final de la página de una empresa
Fuente propia

- **Pantalla con el Formulario**

En la primera versión, habían varios detalles del formulario que no quedaban claros. No existía ninguna explicación previa sobre el carácter público de los datos de la denuncia. Los datos voluntarios parecía que solo los podías añadir en caso de haber sufrido un tipo específico de denuncia. Por último, el consentimiento podía generar falsas expectativas de anonimato a los usuarios, ya que aunque no se mostrase la descripción en la aplicación, cualquier persona con conocimiento sobre Ethereum podría leerla.

Por lo tanto, las modificaciones tratan de solventar estos problemas. Antes de comenzar con el cuestionario en sí, aparece un diálogo que explica para qué se van a utilizar los datos y se recalca la importancia de entender que estos son públicos. En el formulario se han producido 3 cambios:

- Aclarar que cualquier denunciante puede incluir los datos que desee, a excepción de los obligatorios, que debe rellenar sí o sí.
- La relación actual pasa a ser un campo obligatorio para la confección de las gráficas. Se añade una opción de Prefiero no responder, por si el denunciante siente que este dato viola su intimidad.
- Se elimina el consentimiento. Carecía de sentido que aunque en la aplicación no se mostrase la descripción del suceso, se pudiese leer a través de la blockchain. Por lo tanto, todas las descripciones que se introducen en las denuncias pasan a ser visibles.

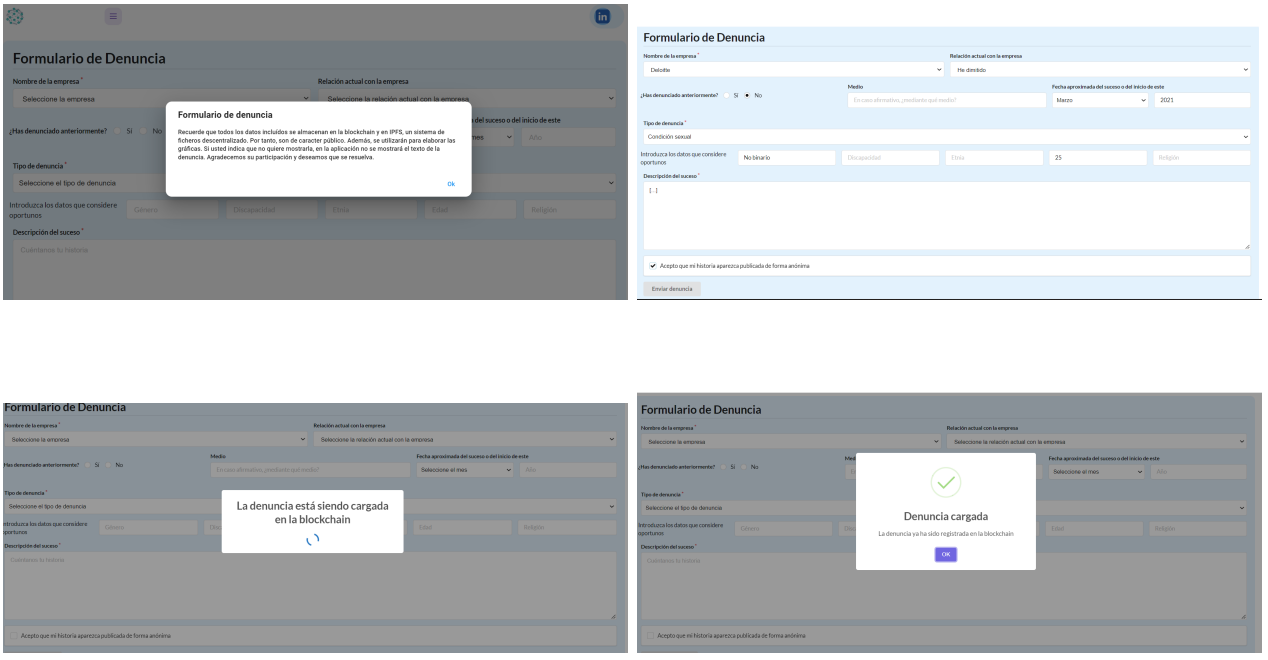


Figura 73: Diseño final del formulario de denuncias
Fuente propia

- **Pantalla con información del proyecto:**

Como ocurrió con el resto de pantallas, tratamos de implantar lo máximo posible de la versión anterior, aunque en este caso el cambio fue mayor, ya que se aprovechó esta pantalla para incluir nuevos elementos informativos. El mayor cambio visual fue la incorporación de una

tarjeta en la que podemos encontrar todas las empresas que podemos denunciar a través de nuestra plataforma, y en la que, haciendo clic en cada una de ellas, puedes acceder a la web de la empresa.

Además, decidimos añadir una idea que ya teníamos durante la primera versión: un botón para donar que nos permitiese obtener un método en el que los usuarios que acceden a nuestra plataforma pueden contribuir con la causa y financiar el coste de las denuncias realizadas. De nuevo, como ocurre con el botón de denunciar, este aparece en la parte superior en un color llamativo, empleando la **ley de Von Restorff** para así reforzar una acción por encima de las demás y conseguir llamar la atención del usuario y hacer más sencilla y rápida su experiencia de donación. Es mucho más fácil conseguir que alguien done si el proceso es rápido (un solo clic) y fácil de visualizar. Para incitar a los usuarios a donar, incluimos un texto sobre ¿Por qué donar? y una tarjeta con el total recaudado.

Asimismo, se mantuvieron las tarjetas de los miembros del equipo, donde pulsando en cada una de ellas podemos acceder a su correspondiente perfil de LinkedIn. Para añadir información sobre el proyecto empleamos la tarjeta que lo describe. Sin embargo, para hacer la plataforma más transparente y que los usuarios puedan conocer de primera mano cómo se ha calculado la reputación de cada empresa, añadimos una nueva tarjeta que lo explica con detalle, aumentando su confianza en nuestra aplicación.

Tras añadir todas las componentes que contiene, nos dimos cuenta de que se usaban muchas imágenes (para cada empresa y cada miembro del equipo) que suponían una carga de memoria para el proyecto. Por ello, se decidió vincular el proyecto con una hoja de cálculo de Google con el fin de guardar en ella las urls de las imágenes [158] que necesitásemos mediante **google-spreadsheet**. Además, las url de las empresas y de los LinkedIn de los miembros también estarían es este sheet. Esta técnica fue aprovechada en todas las partes del proyecto que empleaban dichas imágenes: el carousel y los tops de la pantalla principal, y la cabecera de la pantalla de cada empresa.

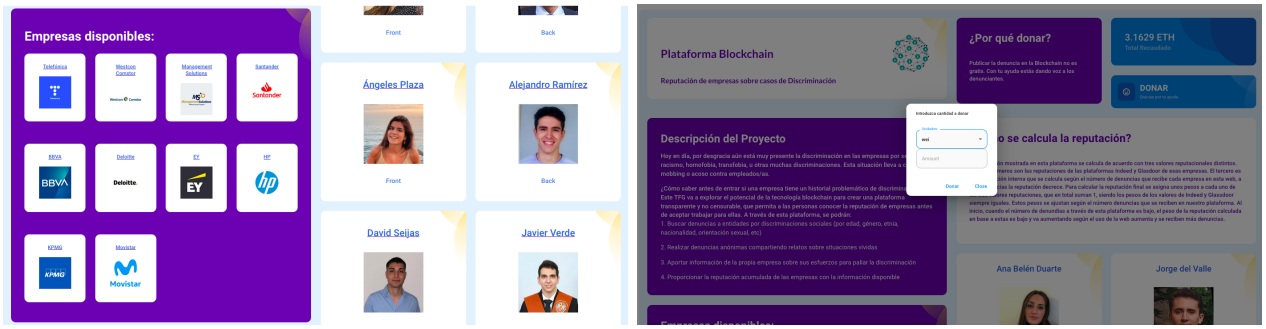
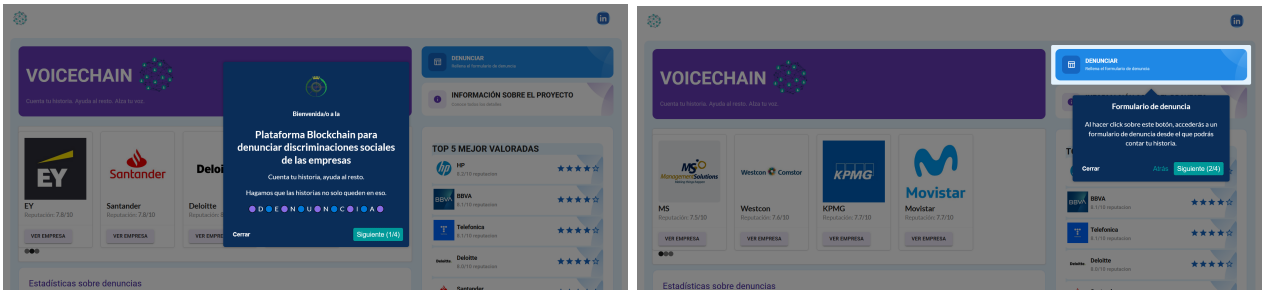


Figura 74: Diseño final de la página de información del proyecto
Fuente propia

- **Onboarding:**
  Además de esto, nos dimos cuenta de que para nosotros el procedimiento a seguir para interactuar con la plataforma nos parecía muy intuitivo porque habíamos trabajado mucho en él, pero quizá para un usuario externo que denuncia por primera vez, o que simplemente quiere visitar la plataforma, podía llegar a ser algo confuso. Por ello, desarrollamos un tour guiado a través de la plataforma, es decir, un onboarding.

  Para el **onboarding**, hicimos una fase de investigación en la que recopilamos una serie de bibliotecas de código abierto que permitían el desarrollo en React. Para decidir cuál de ellas era mejor para nuestra plataforma, buscamos aquella que aportase mayor capacidad para cambiar el estilo y poder ajustarse al diseño de nuestra aplicación. Queríamos un aspecto limpio pero que a la vez fuese funcional y fácil de emplear. La biblioteca elegida fue React Joyride [159].

  En cuanto al diseño, pensamos en emplear una tarjeta inicial que pusiese al usuario en contexto: qué es esta plataforma y qué te permite hacer como usuario. A continuación, queríamos pasos que tuviesen relación con las funcionalidades principales del proyecto: cómo denunciar, cómo ver una empresa y cómo acceder a la página de información del proyecto, donde el usuario podrá realizar una donación y conocer cómo se calcula la reputación. Además, cada uno de estos pasos resalta el componente del que está hablando, dejando el resto de la pantalla en un tono más oscuro, para que el usuario entienda a la perfección a qué elemento se refiere el texto de cada tarjeta.



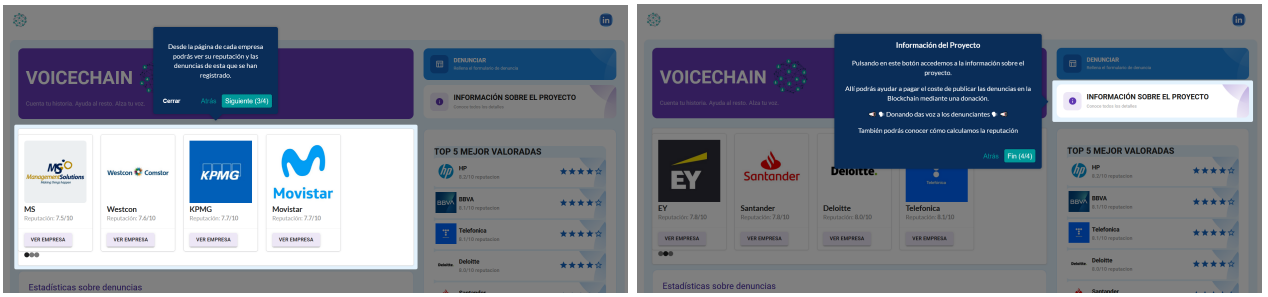


Figura 75: Diseño del onboarding
Fuente propia

Una vez hubimos terminado con el onboarding, nos dimos cuenta de que se hacía muy pesado que cada vez que se recargase la página apareciese. Por ello, pensamos en usar una cookie para guardar la información sobre si el usuario había finalizado alguna vez el onboarding. Sin embargo, como las cookies perduran el tiempo que las definas al crearlas, decidimos usar la variable localStorage [160], [161]. De este modo la información perdura hasta que se limpia la caché e información local, es decir, el onboarding solo aparecerá una vez para cada dispositivo a menos que borremos la variable en cuestión.

8. Trabajo Individual

En este capítulo se detalla el trabajo individual de cada miembro. En algunos casos, la misma tarea fue realizada por varios componentes a través de quedadas o llamadas por Discord. Hemos decidido que en estas situaciones, quedará reflejado el trabajo en aquellos que hayan participado. Además, queremos resaltar que implementar una aplicación web, funcional y estética, requiere mucho más tiempo del que parece con las tareas. Esto se debe a que frecuentemente se presentan errores, que incluso en algunos casos tardaron días en solucionarse.

8.1. Ana Belén Duarte León

Miembro encargado del diseño e implementación del Front-End de la aplicación. En las primeras etapas del proyecto se dedicó a investigar sobre Ethereum, Bitcoin y Blockchain para familiarizarse con esta tecnología puesto que su conocimiento previo sobre la misma era muy escaso, prácticamente inexistente. Realizó un curso de Udemy sobre Blockchain y el desarrollo de DApps desplegadas en Ethereum.

Tras esta primera fase de investigación sobre la tecnología que se iba a usar y, entendiendo los fundamentos principales sobre los que se sostiene, realizó más tarde una investigación sobre el impacto y las implicaciones de la Blockchain en el derecho a libertad de expresión. Además, leyó artículos sobre el potencial impulso que significa la Blockchain para desarrollar plataformas resistentes a la censura y qué problemas presenta relacionados con la seguridad y la gobernanza de las DApps. Más tarde, realizó un análisis de proyectos similares centrándose en plataformas que recogen denuncias sobre casos de discriminación o de colectivos vulnerables. Uno de los principales focos de esta recopilación de información se puso en ver cómo abordaban otros proyectos el acercamiento a la víctima y la recogida de información sobre los hechos denunciados. Un proyecto en el que se basó fue *Qué se sepa*, el estudio sobre violencias sexuales más grande de la historia para entender la investigación teórica previa a un proyecto sobre un tema tan delicado, así como el diseño del cuestionario del mismo y el análisis de resultados posterior.

Una vez asentados los conocimientos sobre el contexto general de la plataforma, desarrolló junto al resto de miembros del equipo un smart contract inicial que representaría el eje principal del proyecto. En este punto, se describieron los requisitos iniciales del mismo: necesidad de almacenar denuncias y creación de un sistema de reputación confiable para los usuarios con el que validar el comportamiento de las empresas. Esta primera versión la implementó en Remix y se solucionaron y solventaron las primeras dudas relacionadas con Solidity y la Blockchain. Este primer contrato era muy sencillo con una lógica muy básica, pero serviría de punto de partida para una posterior versión más sofisticada.

Tras esto, se dedicó a realizar, junto a la otra compañera de Front, el estudio de qué pantallas tendría la aplicación y qué funcionalidades y vistas requería cada una de ellas. Después de hacer esta pequeña pero imprescindible definición, diseñaron los bocetos de las pantallas en un Mockup con la herramienta Balsamiq basándose, como se explica en secciones previas, en los 10 principios del diseño gráfico. El prototipo en Balsamiq sufrió pocas modificaciones y rápidamente se pasó a su implementación en React.

Puesto que el curso de Udemy contenía un tema sobre React y desarrollo del Front-End de un proyecto no fue necesario indagar mucho más, además de que las dudas irían surgiendo a medida que se empezaran a incluir componentes y funcionalidades. En un primera instancia, se decidió

crear un diseño íntegramente propio, partiendo de un "folio en blanco". Así, Ana diseñó el Carousel perteneciente a la página principal, el encabezado y las tarjetas que contendrían la información sobre las empresas. Estos componentes se distribuyeron conforme el Mockup previamente realizado. Posteriormente, se incluyeron componentes con gráficas diseñadas usando la librería *Recharts* para mostrar las estadísticas sobre las denuncias registradas en la plataforma. Se decidió así, que la página principal debía contener como componentes principales: botón para realizar las denuncias, puesto que es la funcionalidad principal, un espacio donde ver las empresas sobre las que se tiene información y gráficas que resuman la información general recopilada por la plataforma en cuanto a número de denuncias, tipología de las mismas y otras cuestiones que se consideraron de interés como el número de denuncias por rango de edad y por género. Estos datos y gráficos ayudarían a tener una visión general sobre los sectores más discriminados y la causa.

Una vez desarrollado el código para la página principal se pasó al diseño del formulario, cuyo proceso dividió en dos partes bien diferenciadas. En primer lugar, investigó y analizó ejemplos de formularios de plataformas similares y propuso al resto del grupo un borrador de formulario de denuncia basado en el formulario prestado por la comisión de derechos humanos del gobierno de Australia para el estudio de discriminación de grupos vulnerables. Junto al resto del equipo se dio forma a las preguntas y se adaptaron al contexto y objetivos de la plataforma. Basándonos en literatura sobre los tipos de discriminación social en los lugares de trabajo, el formulario de denuncias sufriría posteriormente algunas modificaciones. En segundo lugar y haciendo uso de la librería de Material UI se creó el formulario y se incluyó en la pantalla correspondiente.

Tras este trabajo, se tuvo la primera presentación a los tutores que aún estando satisfechos con esta versión como primer acercamiento, se decidió hacer uso de alguna plantilla de la librería **Material UI** para hacer la aplicación más estética. Ana se encargó del acoplamiento del proyecto en la plantilla, así como de la reorganización y adaptación de éste. Organizó la jerarquía de rutas, el índice y las pantallas. Tras esto, comenzó el rediseño y reestructuración de la interfaz gráfica. Los pasos seguidos y/o componentes diseñados en este proceso fueron los siguientes:

- Estructuración básica inicial del proyecto: acoplamiento de la anterior estructura cliente-servidor a la plantilla. Ana realizó la estructura básica del proyecto y su jerarquización, con las rutas convenientes y creó el menú de selección para navegar entre páginas.
- Rediseño y adaptación del encabezado: se eliminaron los componentes que no eran necesarios en el proyecto como la barra de búsqueda y se introdujo el botón para autenticar la cuenta de LinkedIn junto a su funcionalidad. En este punto, se cambiaron también ciertos aspectos de la apariencia del encabezado como el logo y los colores.
- Adaptación del contenedor de empresas: las tarjetas sufrieron pequeñas modificaciones para adaptarse al estilo de la plantilla, fue necesario cambiar los botones, centrarlos y redimensionar las imágenes.
- Diseño de gráficas: usando la librería *Recharts* se diseñaron gráficos de barras donde mostrar la información cuantitativa de las denuncias recogidas por la aplicación. En este aspecto, se trató de personalizar las gráficas al máximo para incluir títulos, leyendas y colores propios, así como una adaptación al tamaño de los datos recibidos. Estas gráficas junto a las diseñadas por Ángeles se combinaron en un componente estilo Grid para que apareciesen juntas y siguiendo una apariencia concreta de tamaño y orden en la página de inicio.
- Adaptación del formulario: de la misma forma el cuestionario se adaptó al estilo de la plantilla usando los botones y colores de la propia librería aunque luego sufriría nuevas modificaciones

por parte de Alejandro para terminar de perfeccionarlo.

- Diseño de Top5 empresas: realizó un nuevo componente para recoger y mostrar cuáles son las 5 empresas con mayor y menor reputación respectivamente. Para el diseño de este componente y basándose en el principio de consistencia externa creo una escala estrellada que se colorea conforme a la reputación de la empresa siguiendo una función que otorga 1 estrella completa cada 2 puntos de reputación, puesto que el máximo es 10.

Finalmente, modificó la pantalla de inicio de acuerdo al feedback y comentarios extraídos de las evaluaciones de usuario.

8.2. Ángeles Plaza Gutiérrez

Miembro del equipo al que se le asignó, junto con Ana, el diseño e implementación del Front-End de la plataforma. Al comienzo, todo el trabajo se basó en investigación. Estudió las tecnologías necesarias para entender cómo funcionaba la Blockchain, así como cuáles eran las tendencias actuales a la hora de realizar plataformas sobre Ethereum, ya que algo totalmente desconocido para ella. Para ampliar conocimientos y a la vez conocer cómo programar tanto en React como en Solidity, completó con éxito un curso teórico-práctico de Udemy en el que se realizaban diversas prácticas que combinaban Back y Front. Tras las primeras reuniones y siguiendo las sugerencias de los tutores, trató de bajar a tierra muchas de las ideas que estaban aún en el aire. Para ello, entre otros, investigó sistemas reputacionales basados en tokens que existiesen en la actualidad y fuesen empleados no solo por DAOs, sino por cualquier organización, para buscar una implementación que cumpliese los requisitos que buscábamos en nuestra plataforma y nos ayudase a preconcebir una idea inicial sobre la que apoyarnos para avanzar.

Asimismo, realizó junto con el resto del equipo la primera versión del **Smart Contract** en Remix, sobre la que se empezaría a desarrollar la plataforma. Fijaron requisitos y estructura.

Llegados a este punto, se reunió con el otro integrante del equipo de Front para realizar el diseño inicial sobre papel, pasando posteriormente estos bocetos a la herramienta de maquetado Balsamiq. En este primer diseño primaba crear una plataforma fácil de usar donde las funcionalidades básicas, denunciar y ver la reputación, estuviesen disponibles para el usuario en pocos clicks. En este Mockup se realizaron decisiones sobre el número de pantallas a emplear, qué elementos tendría cada una y la disposición de estos, siempre teniendo en cuenta los 10 principios de diseño.

Ahora empezaba el trabajo práctico: realizar el código correspondiente para poder visualizar estas pantallas. Para ello, a Ángeles se le asignaron las pantallas de **Empresa** e **Información del TFG**. La pantalla del **Formulario** se realizó de manera conjunta con Ana en esta primera versión. Tras una conversación con Silvia, se dio cuenta de la importancia de generar un formulario en el que nadie pudiese sentirse agredido o interrogado, por lo que se estudiaron los formularios utilizados en otras plataformas similares. En una reunión con Ana se investigaron las posibilidades para crear el formulario y se decidió emplear *semantic-ui*. A continuación, ambas realizaron las modificaciones necesarias para generarlo, adaptando las componentes al boceto creado previamente. Para las preguntas de opción múltiple, como el país, se decidió mostrar una lista con todas las opciones posibles en lugar de un input. De esta manera, la recopilación de datos para las gráficas sería más fácil y veraz.

La pantalla de Empresa contenía, a parte de el encabezado y el botón de denunciar, una tabla con la información de cada denuncia. Para ello, se realizó un estudio sobre las tablas disponibles. *Material table* parecía la librería que ofrecía una tabla más completa, ya que se podía realizar el diseño completo componente a componente y añadir funcionalidades extra. De esta manera, creó una tabla en la que era posible filtrar por orden ascendente o descendente, o introduciendo los datos en cualquier columna. Además, para leer la experiencia del denunciante, existía un botón que estaba deshabilitado si este no había dado permiso. Añadió por último gráficas que mostraban los datos sobre las denuncias a esa empresa. Hizo uso de la librería *Recharts* para crear las gráficas pieChart (gráfica circular), adaptándolas al diseño de la plataforma en cuanto a los colores y el modo de mostrar la leyenda, añadiendo en la parte inferior un resumen sobre la información en términos de porcentajes. La pantalla con la información del proyecto en esta primera versión era muy sencilla: un texto con el objetivo de la plataforma y su compromiso, además de una tarjeta por cada miembro del equipo acompañada de su rol.

Después de la reunión en la que se mostró a los tutores la primera versión del proyecto, se decidió cambiar totalmente el diseño. Para ello, realizó junto a su compañera una investigación sobre las plantillas disponibles y gratuitas: qué ofrecían y cuál de ellas era más compatible para adaptar a la versión actual. Tras una reunión con Ana se decidió la estrategia para transicionar de la versión anterior a la plantilla actual, fijando dónde iban a ubicar cada componente.

En esta versión, Ángeles se encargó de las pantallas de Empresa e Información del TFG, que sufrieron grandes cambios. En el caso de las empresas, necesitaba emplear una tabla totalmente diferente, ya que la anterior no casaba con el nuevo diseño. Decidió crear una tabla muy visual formada por componentes de *Material UI*, donde era fácil de identificar cada tipo de denuncia por mostrar un avatar con la inicial de dicho tipo. En ella, para leer la experiencia, creó un dialog que aparecía al pulsar el botón correspondiente, oscureciendo el resto de la pantalla. El último cambio realmente significativo que aporta a partes iguales funcionalidad y limpieza fue el modo en que se mostraban las gráficas. Creó un componente basado en pestañas donde cada pestaña representaba una gráfica acorde a distintos datos recabados en el formulario, con su título correspondiente. De esta forma, el usuario puede interactuar y decidir qué gráficas observar. Además, el botón para denunciar se hizo más llamativo y estético, y se añadió una tarjeta con la reputación de la empresa.

La pantalla con la información del proyecto mejoró considerablemente respecto a la versión anterior. Como las denuncias costaban ether, suministrado por la cuenta de la plataforma, decidió crear una sección para donaciones, con un botón donde los usuarios pudiesen donar y una tarjeta en la que pudiesen ver el total de ether recaudado con las donaciones. Para motivar a los usuarios, añadió un texto *¿Por qué donar?*; de esta forma, ayudan a la plataforma con un simple clic. Para hacer la plataforma más transparente, incluyó una nueva tarjeta que informa a los usuarios sobre cómo se calcula la reputación, separada de la tarjeta principal con información sobre el proyecto. Finalmente, se incluyeron dos últimos elementos a esta pantalla que sacan su información de un *Google Sheet*, como explicaremos a continuación. Uno de ellos contiene a todas las empresas, con sus imágenes, y es posible hacer clic en cada empresa para acceder a su web. En la otra tabla aparecen imágenes de los miembros del equipo. Pulsando en ellas es posible acceder a su LinkedIn.

Tras percatarse de que en todas las pantallas se empleaban muchas imágenes, una para cada empresa y cada miembro del equipo, estudió un método para deshacerse de ese coste. Decidió emplear una Hoja de cálculo de Google formada por dos *Sheets*, uno con información sobre las empresas y otro sobre los miembros del equipo. Al vincular el código a esta hoja de cálculo es posible ahorrarse el almacenamiento de todas estas imágenes. Para ello, antes tuvo que autenticarse en la *Google Cloud Platform* y crear un proyecto con una cuenta de servicio para obtener los credenciales a usar en el código. A continuación, con la ayuda de Javier, añadió estas credenciales al *context provider* para privatizarlas. Posteriormente, rellenó las celdas de la hoja de cálculo con las url de las imágenes que se querían mostrar, así como otros valores asociados (nombre, link a la web de la empresa...). De esta manera, para añadir una nueva empresa basta con añadir una nueva fila al *sheet* correspondiente. Vinculó dicha hoja de cálculo al código mediante *google-spreadsheet*, un API wrapper de Google Sheets para JavaScript. Investigó cómo enlazar el proyecto con la hoja de cálculo y qué funciones había que emplear para sacar los datos y tenerlos disponibles en el código. Modificó los archivos del proyecto donde aparecían estas imágenes para emplear en su lugar los datos de la hoja de cálculo.

Tras una autoevaluación de la plataforma se dio cuenta de que esta era fácil y funcional, pero podía parecer confusa para alguien que no la conocía. Por ello, decidió realizar un **Tour Virtual u Onboarding**, donde los usuarios que accedían por primera vez a la aplicación podrían entender

su finalidad y cómo usarla. Para ello investigó sobre qué librerías había disponibles y eligió aquella cuyo diseño casaba más con el proyecto: React Joyride. Implementó un onboarding de 4 pasos, redactando ella misma el texto correspondiente y realizando todo el diseño. Para conseguir que al usuario solo le apareciese en la primera visita a la plataforma, relacionó dicho tour con una variable de *local storage*, similar al uso de una cookie.

Por último, realizó una de las pruebas con usuarios: dirigió la sesión siguiendo el guión y tomando anotaciones sobre los comentarios del usuario. Asimismo, diseñó el titulo de la aplicación uniendo las palabras Voice Y Blockchain, para mostrar que nuestra plataforma alza la voz de los denunciantes en cadena, Voicechain.

8.3. Alejandro Ramírez Rodríguez

Miembro encargado del Sistema Descentralizado de la aplicación. En las primeras etapas del proyecto se dedicó a investigar sobre sistemas de reputaciones, de denuncias y descentralizados, especialmente Blockchain. Sus conocimientos iniciales sobre los tema eran escasos, tan solo lo escuchado de pasada en conversaciones con amigos. Por ello, realizó múltiples lecturas. En cuanto a Blockchain, Bitcoin y Ethereum, completó dos cursos de Udemy y las prácticas iniciales que el tutor indicó.

Así adquirió conocimientos suficientes para comenzar a programar un contrato inteligente. Implementó la primera versión del contrato en Remix junto con el resto de participantes del sistema descentralizado. Como ya se ha mencionado anteriormente, este no tenía una gran complejidad, aunque debía funcionar a la perfección. Serviría de trampolín para confeccionar el de la versión definitiva. En cuanto a las dificultades que encontró durante el proceso eran las propias de un novato de la tecnología, por lo que fueron sencillas de solucionar.

Una vez desarrollado el contrato, decidieron compilarlo y desplegarlo desde local. A pesar de ser más sencillo desde Remix, esta aplicación no era la adecuada para proyectos de gran desarrollo. Por ello, tuvo que investigar cómo hacerlo desde local. Decidió utilizar la librería web3j y un provider de truffle que establece la conexión con la red de Ethereum a través de un nodo de Infura. Para poder utilizar dicho nodo, fue necesario crear una cuenta de Infura, así como otra de Metamask para poder financiar todas las llamadas al contrato. Escogieron la red de Rinkeby para desplegar el contrato, ya que obtener ethers ficticios para la cuenta de Metamask en esa red era sencillo.

Implementó los ficheros *compile.js* y *deploy.js*. Estos tienen una mayor complicación que el contrato inteligente inicial, pues requieren un código más complejo y se utilizan múltiples librerías, tanto para la conexión con la red (web3j, truffle) o el manejo de carpetas y ficheros (fs-extra, path). Además, el fichero compile crea una nueva carpeta en la que almacena los contratos compilados en ficheros JSON. Aparecieron problemas a la hora de obtener el objeto compilado, pero gracias a los comentarios de uno de los cursos se pudieron solucionar.

Por último, para finalizar con el trabajo de la parte descentralizada en la primera versión, era necesario establecer la conexión entre el Front y el contrato. Como ya conocía la librería web3j, decidió utilizarla. Además, para poder establecer la conexión de forma correcta, tuvo que entender toda la estructura del Front. Modificó todos los ficheros en los que se utilizaban datos de prueba e introdujo todas las llamadas. Como las llamadas son asíncronas, tuvo que aprender a manejar promesas en Javascript para que todo funcionase correctamente. Consiguió solucionar todos los problemas.

Además, mientras añadía la conexión, necesitó cambiar la implementación del cuestionario para emitir una denuncia, implementado en el fichero *form.js*. Introdujo un hook especializado en formularios, *useForm()*, que facilitaba la obtención de los datos y los checkeos de errores. La llamada al contrato para almacenar la denuncia tarda unos 15 segundos en ser efectiva (lo que tarda en aparecer la transacción en un bloque minado), por lo que incluyó junto con David Seijas diálogos que informan al usuario de la situación de la transacción, para que no crea que hayan surgido problemas.

Por último, ya que la carga de trabajo de la capa descentralizada en esta versión era menor que la del resto, decidió echar una mano al equipo de Front en errores de la página principal y de la página de cada empresa. Colaboró en la tabla de denuncias para mostrar el texto, utilizando elementos y propiedades de la librería *semantic ui*. También se encargó del enrutamiento dinámico

a las páginas de las empresas, mediante la librería *react-router-dom*.

Tras la presentación de la versión inicial con los tutores, todo el equipo se puso manos a las obras con la versión definitiva. En cuanto al sistema descentralizado, se encargó de tres tareas principales: modificar los smart contracts, tokens para la reputación y una métrica consistente que introduzca la reputación externa.

Para los tokens, investigó sobre posibles usos de tokens reputacionales, así como de los no transferibles. Encontró un par de propuestas que se adecuaban a los requisitos del proyecto y decidió utilizar una de ellas: EIP-4794. Sin embargo, debido a su volatilidad, pues todavía no es aceptado entre la comunidad, decidió crear otra solución con unos tokens reconocidos: ERC-20 o estándar de tokens fungibles.

Confeccionó los contratos necesarios para manejar los tokens, así como los dos contratos principales de cada solución. Como David Seijas era el encargado de la sección IPFS, tuvo que comunicarse con él para modificar los contratos principales. Añadieron nuevas funciones y variables de estado para rematar la lógica de la aplicación basada en Blockchain. Además, se mejoró la implementación tratando de ahorrar al máximo el consumo de gas.

Respecto a Blockchain, el último cambio que realizó fue sobre los ficheros *compile.js* y *deploy.js*. Se adaptaron para poder compilar y desplegar múltiples contratos, así como seguir las rutas de los imports de los contratos. Debido a la escasa literatura sobre este proceso, los problemas que aparecieron fueron complicados de solucionar. Tuvo que modificar ficheros de configuraciones de VSCode, de la extensión empleada y del compilador de Solidity descargado.

La tercera tarea de la que era encargado consistía en elaborar una métrica para unir los tokens con la reputación externa, que se obtendría de Glassdoor e Indeed. Se comunicó con ellos vía correo electrónico para el acceso a sus APIs. Recibió una respuesta negativa y otros compañeros se encargaron del scraping para obtener dichos datos. Ideó la métrica junto con Jorge del Valle, la cual está basada en una media ponderada cuyos pesos modifican su valor en función de cuánto ha crecido la aplicación. Además, para poder utilizar esta fórmula, los tokens deben reflejar un valor numérico entre 0 y 10. Para ello creó otra ecuación que tiene en cuenta la proporción de tokens con respecto al resto de empresas y la propia cantidad de tokens que posee una empresa.

Una vez desarrollado y desplegados los contratos y de forma análoga a la versión inicial, se encargó de la conexión entre el Front y los contratos. Modificó todos los ficheros necesarios, para ello se introdujo el uso de una nueva librería, *ethers*, pues mucha gente la recomendaba y web3j tenía algunas funciones obsoletas. Este proceso fue el más complejo debido a los múltiples fallos que surgieron en la conexión entre el Front, IPFS y el contrato. A pesar de haber aprendido el manejo de las promesas en la versión inicial, fue complicado anticipar los errores. Para solucionarlos tuvo que entender a la perfección el ciclo de vida de los componentes y el funcionamiento de las promesas. Al final consiguió que todo funcionara correctamente.

En cuanto a la parte de Front modificó el formulario para que al principio el usuario se encontrase con una pantalla que le explicara para qué se van a usar sus datos. También aclaró algunas preguntas que había en el propio cuestionario y eliminó el consentimiento. Este dato se utilizaba para elaborar una de las gráficas. Por ello, tuvo que realizar múltiples cambios para mostrar la relación actual con la empresa en vez del consentimiento: llamadas a la base de datos; capturar los datos de IPFS; diccionarios para transformar los datos procedentes del servidor e IPFS. Además echó un

cable en los tops de la página principal y en la tabla de denuncias de cada empresa. Por último, se encargó de mostrar la cantidad de ethers donadas. Como no existe el tipo float en Ethereum, dicha cantidad se almacena como un string en el contrato principal y se convierte a float en el Front para tratarlo.

Además, después de todo, fue uno de los encargados de la evaluación con usuarios. Elaboró el guión inicial de los moderadores y la lista de tareas de las pruebas. Realizó una evaluación a un usuario. Dirigió la sesión siguiendo el guión y tomó notas de todo lo que sucedió, así como de las opiniones del participante.

8.4. David Seijas Pérez

Miembro encargado del Sistema Descentralizado de la aplicación. Al comienzo del proyecto tuvo que investigar, al igual que los demás miembros del equipo, sobre las nuevas tecnologías que iban a estar involucradas. Además de las diversas prácticas introductorias al TFG explicadas en la presente memoria, se documentó sobre proyectos similares, cómo se podía implementar el sistema de reputación y realizar el formulario de denuncia de la mejor manera posible. Sin embargo, lo más importante al inicio era conocer con el mayor detalle posible las nuevas tecnologías como Blockchain y Ethereum. Para ellos realizó dos cursos en Udemy para conocer las bases y el funcionamiento de estas. Además, completó un curso adicional sobre el lenguaje de programación de Ethereum, Solidity, ya que iba a ser uno de los encargados de la parte de Back-End y de la implementación del Smart Contract sobre el que se basa el proyecto.

Una vez superada esta fase de investigación, el equipo se propuso realizar en las primeras semanas una versión funcional sobre la que apoyarse en el inicio con la finalidad de ir avanzando. David fue el encargado de la realización del primer Smart Contract en Remix, el cual era muy sencillo y su función principal eran la de añadir una denuncia que almacenara la información especificada por el usuario en la red Ethereum. Cabe decir que, por facilidad y lógica, el equipo de la parte descentralizada decidió usar la red de Rinkeby, pues es una red de prueba de Ethereum en la que no se gastan ethers reales por desplegar contrato.

Una vez tuvo el Smart Contract implementado y probado su funcionamiento, se encargó de trasladar el contrato a local, desarrollando los archivos *deploy.js* y *compile.js* con los conocimientos adquiridos en los cursos previos al proyecto. Sin embargo, desplegar un contrato desde local no era tan sencillo, se necesitaba un portal de acceso a la red de Rinkeby. Con la investigación realizada al inicio averiguó las distintas formas de acceder a la red a través de un nodo: usando un nodo propio o usar un nodo provisto por Infura. Además, se necesitaba hacer uso de Metamask para desplegar el contrato desde ella pagando con ethers de la cuenta. Por ello, creó cuentas tanto en Metamask como en Infura para el proyecto y se encargó de obtener ethers gratuitos para Metamask a través de un proceso de obtención de ethers sin coste en Rinkeby aprendido en los cursos. Con estas cuentas creadas y gracias a la librería web3j, aprendida también durante los cursos y que nos da acceso a la red Ethereum, desplegó el contrato y obtuvo la dirección de este en la red, pues iba a ser necesario para desarrollar la aplicación.

Desplegado el Smart Contract, comenzó a desarrollar toda la conexión entre Front-End y Back-End. Para ello, primero modificó todos los archivos involucrados en el formulario para poder extraer los datos reales introducidos por el usuario, así como datos como la fecha del momento de la denuncia. Estos fueron cargados en la red de Rinkeby almacenados en una estructura de datos específica para la denuncia. El siguiente paso fue completar los archivos encargados de la muestra de las empresas, las denuncias presentadas sobre estas y todas las reputaciones, cambiando los datos de prueba usados por los datos reales, previamente cargados, extraídos de nuestra plataforma. Como en esta parte se enlazaba con el Front-End, hecha con React y Javascript, tuvo que aprender lo necesario sobre estas tecnologías, sobre todo el funcionamiento de las llamadas asíncronas y el manejo de promesas.

Una vez esto fue realizado con éxito, ayudó a completar la parte de Front-End, cuya extensión era mayor al inicio del proyecto que otras. Aportó en la estética de la interfaz, para lo cual tuvo que investigar y aprender sobre la librería *semantic-ui*. Lo más destacado que aportó, además de la solución de pequeños errores, fue introducir diálogos de interacción con el usuario, destacando lo

hecho en la parte del formulario, donde introdujo indicaciones y avisos a los usuarios, por ejemplo durante la espera a que la denuncia sea cargada en Rinkeby.

Tras todo este trabajo, tuvo lugar la presentación de la primera versión con los tutores. De esta se sacó en claro que en la parte descentralizada era necesaria la inclusión de la tecnología IPFS y la introducción de tokens reputacionales para medir el sistema de reputación de la empresa. David fue el encargado de incluir la tecnología IPFS.

De los cursos e información recabada al inicio del proyecto, David conoció esta tecnología y su uso, pero nada sobre la implementación de ella en un proyecto. La idea era clara, almacenar la denuncia en el sistema de archivos distribuidos de IPFS en lugar del Smart Contract, donde solo almacenaríamos el hash del archivo de la denuncia para poder acceder posteriormente a ella. Para ello, lo primero que hizo fue modificar el Smart Contract. Cambió las estructuras de los datos usados y la función específica en la que se incluía la denuncia al contrato. Además, añadió funciones nuevas para extraer los hashes de las denuncias almacenadas previamente. Por otra parte, junto con Alejandro Ramírez, decidieron modificar algunas funcionalidades para mejorar el consumo de gas, una vez tenían más conocimiento gracias a cursar la optativa *Introducción a la tecnología Blockchain y Smart Contract*.

Después de esto, valoró cómo almacenar la denuncia. Decidió guardarla como un arhivo JSON, creado en el momento justo del envío del formulario de denuncia por el usuario, que guardaba toda la información necesaria de esta denuncia. Averiguó que para poder acceder a IPFS y almacenar un archivo había una forma sencilla a través de Infura. Investigó sobre la librería *ipfs-http-client* necesaria para la implementación de esta funcionalidad. Aprendió que gracias a esta librería podíamos crear una instancia que manejara una API ofrecida por Infura para controlar el nodo de esta y acceder a IPFS. Gracias al método *add* ofrecido fue capaz de almacenar una denuncia como una cadena de caracteres que componía el archivo JSON de la denuncia. Además, este devolvía el hash donde se localizaba la denuncia, enviada y almacenada en el Smart Contract. Por último, averiguó una forma de extraer todas las denuncias a través de los hashes almacenados en el contrato. Sin embargo, estas se devolvían codificadas en un array de enteros que se encargó de transformar para recuperar el archivo JSON original y así poder manejar fácilmente los datos necesarios.

Finalmente, incorporada esta nueva tecnología, se encargó de lanzar el nuevo contrato otra vez. Una vez este fue desplegado tuvo que enlazar la parte de Front-End con la de Back-End, pues toda la interfaz fue cambiada tras la reunión con los tutores. Para ello tuvo que investigar y aprender sobre las librería *ethers.js* que iba a sustituir a *web3.js*, pues según se informó su popularidad entre los desarrolladores estaba creciendo por las numerosas ventajas que ofrecía. Su tarea principal en este momento fue, con la información extraída del Smart Contract e IPFS, mostrar todas las gráficas, filtrando los datos necesarios para cada una de las empresas. Además, con los nuevos datos obtenidos del scraping y el número de denuncias obtenido del contrato era necesario calcular la reputación de cada compañía. Para ello, junto con otros compañeros, implementó un sistema en el que el peso de las denuncias presentadas y el peso de los datos del scraping varían según el número de denuncias que haya a través de nuestra plataforma. De esta forma, cuantas más denuncias y más grande sea nuestro proyecto, más peso tendrá el número de denuncias recibidas por una empresa.

De nuevo y para concluir terminar de desarrollar la plataforma, ayudó a la parte del Front-End a solucionar y ultimar pequeños detalles para lo que tuvo que investigar ligeramente a cerca de la librería *material-ui*, que había sustituido en su gran mayoría a la anterior *semantic-ui*.

Además, después de todo, fue uno de los encargados de la evaluación con usuarios. Ayudó en la creación del cuestionario final que debía ser rellenado después de realizar la prueba con los usuarios para que contaras sus sensaciones y opiniones sobre utilidad, manejo y facilidad de la plataforma. Además, realizó una de las pruebas de evaluación a uno de los usuarios.

8.5. Jorge del Valle Vázquez

Miembro encargado del Sistema Descentralizado de la aplicación. Al comienzo del proyecto realizó las mismas tareas que el resto de compañeros del equipo: investigar sobré Blockchain y ver cómo encajar esta potente tecnología en nuestro proyecto. Para ello, se documentó a partir de varios libros y artículos. También completó las prácticas iniciales, descritas en la fase de investigación de este documento. A su vez, siguió un curso de React-Redux, que aunque en el futuro no fuera su objeto de trabajo, le sirviera para tender puentes entre distintas partes del proyecto

Al comienzo de la fase de diseño de la aplicación, desarrolló el primer contrato inteligente junto con el resto de compañeros del equipo del sistema descentralizado. Como ya se ha mencionado anteriormente, esta versión era muy elemental, meramente funcional. El contrato se encargaba de almacenar las denuncias que recibían cada empresa y actualizar el sencillo sistema de reputaciones. Sin embargo, gracias a su sencillez, permitió una agradable aclimatación del equipo con Ethereum, algo novedoso para todos.

A parte de la confección del contrato inteligente, en la versión inicial de la aplicación se encargó de analizar posibles métodos para financiar el proyecto. De entre todos los existentes, frecuentemente en las DApps se promueve el crowfunding, vía monedas virtuales como los ethers. Este es el que mejor se acopla a la aplicación , pues el contrato principal es el que debía recibir las donaciones y así se consigue de forma directa.

Investigó como lo llevan a cabo las DApps, y observó que la mayoría se efectúan a través de Metamask. Así pues, utilizó la librería *ethers* para conectar la cuenta con el Front ofreciendo la posibilidad de donar a cualquier usuario que tenga una cartera de *Metamask*. Aprovechó que esta tecnología inyecta su propio provider en la web, por lo que observando una variable global de window es posible saber si un usuario tiene cuenta o no. Además, tuvo que modificar el fichero *DonationCard.js* para incluir un botón de donar que permitiera escoger la cantidad deseada. Por lo tanto, en este proceso estableció una conexión entre la red de Ethereum y nuestro Front. Dicha conexión es externa al contrato principal, como se muestra en la arquitectura.

En lo que se refiere a esta primera versión funcional, se se dedicó a investigar posibles tecnologías a incorporar, algunas recomendadas en primera instancia por el tutor, como zk-SNARKs que busca probar la autenticidad de los usuarios bajo el concepto de prueba de conocimiento cero. No obstante, el equipo se decantó por la autenticación por LinkedIn que luego implementaría Javier. De todas maneras, tras la reunión con los tutores para mostrarles el prototipo, se centró en aportar soluciones externas al sistema descentralizado, otorgándole así una mayor robustez y consistencia.

El primer conflicto que se presentó fue la simpleza del sistema de reputaciones. Este no tenía en cuenta que al principio todas las empresas poseían una reputación máxima. Por lo tanto, se plantearon múltiples ideas para solucionarlo y al final se escogió obtener valores reputacionales de dos fuentes externas: GlassDoor e Indeed. Contactó con ambas empresas para poder utilizar su API, pero recibió un no por respuesta. Así pues, para solucionarlo implementó un script de Python junto con Javier Verde, que extraía las reputaciones del propio código html, proceso conocido como web scraper. Se planteó recabar toda información sobre toda empresa disponible en cada plataforma, pero se acabó decidiendo por solo almacenar las valoraciones de las empresas del proyecto, debido a que por el momento era innecesario guardar información adicional y llevaría demasiado tiempo. Estos valores los guardó sobre la colección chartData con los formatos y esquemas establecidos por Javier Verde.

Igualmente, era necesario dar credibilidad a estas valoraciones. Se tenía dar peso a la reputación interna que representa el sistema de tokens, pero en su investigación descubrió que los esquemas basados en la Estadística más empleados, como el sistema de reputación Beta, requerían de condiciones que nuestro sistema no cumplía. Entre otros sistemas de reputación no existe alguno cuya reputación y cantidad de tokens sigan una relación inversamente proporcional. Por consiguiente, poniendo en conjunto los resultados de la investigación con Alejandro Ramírez que realizaba el sistema de tokens, ideó una métrica basada en una media ponderada entre las reputaciones externas e interna cuyos pesos modifican su valor en función de cuánto ha crecido la aplicación. Para cuantificar la magnitud de las denuncias a una empresa se relaciona la cantidad de tokens de la propia empresa para con el resto de empresas. Se definieron así un conjunto de funciones que ofrecerán simultáneamente buen comportamiento y credibilidad.

Uno de los problemas con los que se encontró tras la reunión con los tutores, fue la posibilidad de restringir el género a un conjunto de opciones. Se decidió no hacerlo, pero era necesario una mínima clasificación para la gestión de las gráficas informativas. Por ello, desarrolló un pequeño sistema de reconocimiento de palabras con OpenAi[162] por procesamiento del lenguaje natural, que corrigiera errores gramaticales para una correcta clasificación del género, en principio entre los 37 géneros que actualmente se reconocen en España, pero permitiendo también cualquiera que no quede recogido entre los 37. Sin embargo, durante el desarrollo cambió la política de la API que restringió el número de llamadas gratuitas.

En última instancia, fue encargado de realizar parte de la evaluación con usuarios. Entre las actividades que realizó está seleccionar participantes para posteriormente someter a evaluación la realización de una serie de tareas que dieran lugar a un conjunto de resultados que establecieran objetivos de mejora.

8.6. Javier Verde Marín

Miembro encargado del Sistema Centralizado de la aplicación. En las primeras etapas del proyecto se buscaba la forma de autenticar a los usuarios sin tener que almacenar sus datos, de forma que se pudiera garantizar la privacidad y anonimato. Por tanto, se dedicó a investigar distintas formas de identificación que cumplieran con esto además de garantizar en la medida de lo posible que el usuario pertenezca al entorno laboral. Tras la investigación, decidió que el inicio de sesión mediante LinkedIn podría ser una buena opción al ser una red social de ámbito empresarial.

Creó una cuenta de empresa junto con una aplicación en LinkedIn Developers y solicitó el acceso al servicio de API para autenticación. No obstante, para poder efectuar la funcionalidad, era necesaria la creación de un servidor que pudiera incorporarla. Para ello, creo una aplicación de Node.js y montó un servidor mediante Express.js capaz de responder a las peticiones del cliente. Buscando una forma de autenticar al usuario mediante el uso de la API y el servidor, encontró Passport.js; un middleware que como se ha mencionado con anterioridad, permite la realización de login mediante una gran variedad de estrategias.

Escogió la estrategia para LinkedIn basada en OAuth2 y estudió la información de perfil que devolvía la API mediante esta. A raíz de esto surgió la necesidad de crear una base de datos para poder almacenar a los usuarios. Evaluó y comparó el uso de bases de datos relacionales y no relacionales, decantándose por estas últimas; en concreto, MongoDB. De forma que pudiera ser accedida por cualquier usuario que probara la aplicación sin tener que instalar software adicional, creó una base de datos en Mongo Atlas y una cuenta de administrador para el grupo del proyecto cuyas credenciales se utilizarían en la aplicación de Node.js.

Para almacenar a los usuarios, creó la colección userInfo y se encargó de diseñar el formato con el que posteriormente construiría el esquema y modelo para enviarlos a la base de datos. Una vez listos el servidor y base de datos, implementó la lógica de autenticación con el apoyo de las funciones que pone a disposición Passport.js así como los endpoints con los que el cliente puede acceder a la funcionalidad de login y obtención del nombre de usuario. A su vez, en el lado del cliente implementó un proveedor de contexto definido en *contextProvider.js* que realiza la petición de obtención de usuario y las funciones del botón de login/logout contenidas en *userFunctions.js* que se comunican también con el servidor. En adición, el proveedor de contexto sirve para poder comunicar el nombre de usuario a las distintas páginas de la aplicación web. Igualmente, junto al resto del equipo, colaboró en la definición de requisitos y estructura del primer smart contract creado; además de ayudar al equipo encargado del sistema descentralizado en el desarrollo del sistema de donaciones y la primera versión de la función encargada de firmar las transacciones que conforman las denuncias para que el usuario no sea responsable de la realización del pago.

Posteriormente, al aparecer la necesidad de almacenar los datos utilizados para las gráficas y las valoraciones que constituyen las métricas externas, creó las colecciones chartData y companyMetrics. Se encargó de diseñar el formato de los datos y construyó sus respectivos esquemas y modelos de la misma forma que con los usuarios. A continuación diseñó y habilitó los endpoints que permiten la obtención e inserción de datos en chartData y companyMetrics así como la función que inicializa chartData en caso de que no haya ninguna denuncia.

Al estar lista la colección que almacena las métricas externas, intentó ponerse en contacto con varias platformas como Glassdoor, Indeed y Comparably. Ante la falta de respuesta de estas, se encargó de buscar en Indeed y Glassdoor las páginas de las distintas empresas que se han utilizado

en el proyecto y el nombre de los elementos HTML que contienen la valoración de las empresas. Con el objetivo de almacenar esas valoraciones, desarrolló un script de Python junto a Jorge del Valle que funciona a modo de web scraper, recorriendo las distintas urls de cada empresa en las dos plataformas. El script se encarga de añadir las valoraciones de cada empresa a un array y almacenarlos en la colección chartData. De forma que se pueda mantener la información lo más actualizada posible, configuró el servidor mediante un planificador de tareas para que se ejecutara automáticamente en una consola de Python creada a modo de proceso hijo diariamente.

Asimismo, decidió crear una documentación para los distintos endpoints del servidor ya que había aumentado el número de funcionalidades. Para ello, incluyó Swagger a la lista de herramientas utilizadas ya que se puede servir el documento en un endpoint. Otro factor por el que se decidió su utilización es por la capacidad de probar las distintas rutas dentro del propio documento por lo que mejora considerablemente el testing en el servidor, a diferencia de tener un documento a modo de fichero de texto. La documentación se construye a base de las anotaciones escritas mediante comentarios encima de cada uno de los endpoints. Mediante estas anotaciones dividió los endpoints en categorías y definió los tipos de datos que usan o devuelven además de contener ejemplos de body y respuesta.

Acercándose el final del desarrollo del proyecto surgió la preocupación por la tolerancia y recuperación ante fallos por parte del servidor. Para poder cubrir la mayor cantidad de escenarios posibles, realizó una serie de actualizaciones.

Actualizó el servidor para que las sesiones de usuarios que hayan realizado login se pudieran almacenar en una colección generada automáticamente por la librería connect-mongo para que pudiera recuperarse en caso de error. Mediante esta implementación, las sesiones solo desaparecerán si el usuario ha cerrado sesión manualmente o si se ha expirado el ttl de la sesión.

Adicionalmente, generó un Dockerfile para el cliente en caso de que pudiera haber problemas de compatibilidad entre la aplicación y los sistemas de los usuarios que la prueban y otro para el servidor con el fin de poder generar copias de este. Investigó formas de incorporar balanceo de carga con la intención de que pudieran distribuirse las peticiones y que otros servidores pudieran ocuparse de las peticiones recibidas por otro servidor en caso de que caída hasta que encontró la imagen de Nginx en el Docker Hub. Creó un fichero de configuración para Nginx y un Dockerfile que utiliza la imagen encontrada y le aplica el fichero.

Complementariamente, diseñó un archivo Docker Compose que instruye la creación de tres contenedores a raíz de la imagen del servidor que se construye con su Dockerfile, cada uno escuchando en puertos distintos de la máquina sobre la que se ejecuta. También crea un contenedor del balanceador de carga de Nginx que escucha sobre el puerto en el que escucharía el servidor en caso de que no se utilice Docker.

Todos los contenedores se configuraron de modo que se intenten reiniciar en caso de se produzca un error mientras que por otro lado, el balanceador de carga está diseñado de forma que pueda enviar las peticiones a otro servidor en caso de que uno no responda.

Posteriormente realizó pruebas para asegurarse del correcto funcionamiento y comportamiento del sistema centralizado además de brindar apoyo al resto del equipo en la detección y solución de errores presentes en la aplicación a nivel global. Algunas de las pruebas llevadas a cabo son:

- Levantar los servidores y eliminar uno de ellos durante un proceso de login para comprobar si pasados 2 segundos se ocupa otro.
- Cerrar sesión y verificar que realmente se elimina de la base de datos.
- Iniciar sesión en diferentes navegadores de forma que se confirme que solo se almacenan aquellas sesiones producto de un login.
- Testing de los endpoints mediante el documento de Swagger.

Finalmente, fue de uno de los encargados de la evaluación con usuarios, realizando una de las entrevistas y analizando las interacciones del usuario con la aplicación y sus respectivos comentarios para luego poder aplicarse mejoras.

9. Evaluación con usuarios

9.1. Propósitos y objetivos de la evaluación

Para realizar la evaluación con los usuarios, es esencial preestablecer un plan de acción. Esto ayuda a reducir costes y centrar la atención en los objetivos principales. Una vez desarrollado el prototipo final de la plataforma se realizarán evaluaciones con distintos usuarios elegidos convenientemente según ciertos aspectos y requisitos para que interactúen con el prototipo, comprobando si son capaces de ejecutar todas las funcionalidades ofertadas de forma intuitiva y sin errores: poner una denuncia, consultar la reputación de una empresa, leer las denuncias públicas de una empresa, determinar las empresas mejor valoradas, determinar la empresa con peor reputación y examinar los datos de denuncias por tipología, rango de edad y género. Con esta evaluación se pretende comprobar la experiencia de los usuarios, es decir, si son capaces de interactuar con la interfaz de forma intuitiva, sencilla y agradable. Así, se determinarán con mayor precisión que objetivos se han alcanzado, cuáles no y cuáles son las posibles mejoras potenciales de cara al futuro. Al tratarse de una plataforma cuyo fin es atender casos de discriminación en el ámbito laboral, uno de los principales objetivos es que la interposición de la denuncia se realice de forma satisfactoria por todos los usuarios.

9.2. Preguntas de investigación

Una vez establecidos los objetivos del test, vamos a reformularlos generando preguntas de investigación que deben ser respondidas por la evaluación. Recordemos que estas preguntas han de ser específicas, precisas y medibles u observables.

- ¿Es intuitivo para el usuario realizar las acciones principales?
- ¿Entiende el usuario la plataforma al entrar por primera vez?
- ¿Valoran favorablemente los usuarios el onboarding inicial?
- ¿Se pueden ejecutar las funcionalidades más importantes de forma intuitiva?
- ¿Se puede acceder a ellas de forma rápida, es decir, en pocos clics?
- ¿Es cómoda la navegación entre pantallas? ¿Está bien cohesionada?
- ¿Dan lugar a errores de interpretación o de missclicks los iconos y botones?
- ¿En algún momento el usuario no ha sido capaz de encontrar alguna funcionalidad y/o información?
- ¿Confía el usuario en la plataforma?
- ¿Utilizaría el usuario la plataforma para denunciar a una empresa?
- ¿Desconfía de la tecnología (Blockchain) que hay tras ella?

Podemos afirmar que todas estas preguntas son específicas y precisas, de hecho podemos responder con un simple sí/no. En algunas de ellas pueden existir diferentes respuestas en función de cada acción principal. En su mayoría son observables, y algunas son medibles, por ejemplo, contabilizando cuántos botones o iconos pulsa erróneamente el usuario y como se repone de los fallos.

9.3. Requisitos que deben cumplir los participantes

La aplicación está centrada en el ámbito laboral. Es por ello que todos los participantes deben estar trabajando, o haberlo hecho en alguna empresa. Además, la mayoría debe tener LinkedIn para poder emitir una denuncia. No es necesario que hayan experimentado situaciones denunciables, aunque en dicho caso, la evaluación será más enriquecedora. Tampoco son necesarios más requisitos para los participantes, pues la aplicación está enfocada para cualquier trabajador o exempleado que quiera dar voz a su problema, o simplemente analizar la reputación de las empresas, así como las denuncias que hayan recibido.

9.4. Descripción del diseño experimental

Este proceso describe las tareas a realizar, el entorno y las herramientas que vamos a emplear, las tareas del moderador, la identificación de datos a recolectar y la metodología de análisis de datos.

9.4.1. Tareas a realizar

Debemos especificar las tareas a realizar en las pruebas guiadas de forma que se cubran todas las funcionalidades que ofrece la interfaz. Las evaluaciones tendrán una duración de unos 7-8 minutos en los que los usuarios seguirán una lista de acciones a realizar y preguntas que responder según los datos que ofrece la plataforma. Se realizaron un total de 4 evaluaciones formadas por personas que aúnan los requisitos especificados previamente. La lista de tareas a realizar es la siguiente:

- Seguir el onboarding inicial para aprender los elementos básicos de la aplicación.
- Visualizar la página principal y comprender la información que se proyecta en ella: reputaciones, top de mejores y peores empresas, gráficas y sus significados, presupuesto del proyecto...
- Analizar los datos de una empresa específicamente (por ejemplo Telefónica). Leer las denuncias y de nuevo comprender las gráficas.
- Conectarse con su usuario de LinkedIn para poner una denuncia. (Si no tiene, la evaluación será exclusivamente para observar si la aplicación es intuitiva y los datos que se muestran en ella son entendibles).
- Denunciar a HP por mobbing. Añadir los datos que el participante desee. Preguntar si ha entendido bien que sus datos vayan a ser públicos.
- Leer la información del proyecto: creadores, motivación del mismo, presupuesto... Finalizar la evaluación con una donación.

Para que la evaluación sea más productiva, los participantes deben comentar sus opiniones respecto a la aplicación mientras ejecutan cada tarea. Además, el moderador debe tomar notas del proceso: dificultades y en qué tarea se han dado, opiniones negativas de los participantes, opiniones positivas... Es importante que en el resultado de la evaluación quede reflejado si los participantes se han sentido cómodos con el proceso de denunciar. En caso de que el usuario se quede estancado en alguna tarea, el moderador debe ayudarle.

9.5. Entorno y herramientas

La evaluación debe ser un proceso sencillo y agradable para el participante, para tratar de obtener la máxima información posible. Escogemos un entorno en el que cada uno se sienta cómodo.

En cuanto a las herramientas, la evaluación se hará desde un ordenador que tenga descargado el proyecto. Para evitar molestias en el participante, el moderador deberá facilitar el ordenador con todo listo para ejecutar. Además, es necesario conexión a Internet para conectarse a LinkedIn y donar.

9.6. Búsqueda y selección de participantes

Como hemos especificado previamente, la búsqueda de participantes la realizamos para ajustarnos con los requisitos que estos debían cumplir. Es por ello que cada uno de los miembros que componemos el equipo hizo un análisis de las personas de su círculo cercano que trabajan en empresas privadas y/o estaban realizando prácticas en estas.

Siguiendo estas pautas, se eligieron 5 personas distribuido de la siguiente forma:

- Una persona de 23 años que se encuentra realizando prácticas en una empresa privada y quiere ver la reputación de su empresa por si decide quedarse.
- Una persona de entre 25 y 30 años que, actualmente, está trabajando en una empresa privada y estaría interesada en denunciar si sucediese un caso discriminatorio.
- Una persona de entre 25 y 30 años que, actualmente, están trabajando en una empresa privada y busca cambiar de empresa porque no está cómodo con el ambiente de la suya.
- Una persona de entre 40 y 60 años que tenga interés en denunciar.
- Una persona de entre 40 y 60 años que le interesaría ver la reputación de la empresa en que trabaja

9.7. Materiales creados para la evaluación

Antes de comenzar las evaluaciones con los 5 usuarios seleccionados, es recomendable una preparación previa de los materiales y documentos necesarios. Algunos de ellos ya se han expuesto anteriormente en la presente memoria, como el prototipo a evaluar a través de capturas de la interfaz y las tareas que han de ejecutar los usuarios. El resto de documentos que confeccionan la preparación se presentan a continuación.

9.7.1. Screening Inicial

9.7.2. Guión para la orientación

Al inicio de la prueba, el moderador explicará al usuario en qué consistirá, describiendo brevemente la aplicación. Además, le dará algunos consejos útiles para el desarrollo de la interacción. Cada moderador lo hará a su forma, aunque se basarán en un guión común para todos. El texto en cuestión es el siguiente:

Buenos días. Usted ha sido escogido para realizar una prueba de evaluación de nuestro proyecto del TFG. Este es una aplicación centrada en el ámbito laboral, que permite a los usuarios denunciar situaciones de discriminación que hayan sufrido en una empresa. Además, es posible observar las

reputaciones de cada compañía, así como todas las denuncias que han recibido.

Antes de comenzar la evaluación, deberás rellenar un pequeño formulario para conocer su nivel tecnológico, su afinidad con la tecnología Blockchain y su ambiente laboral.

Esta prueba consistirá en la realización de 6 tareas en la aplicación, simulando el comportamiento real de cualquier usuario. La página trata de ser lo más intuitiva posible, con un breve tutorial al comienzo para familiarizarse con ella. Se irán enunciando una a una las tareas, cuando la anterior haya sido superada. Es recomendable que vaya diciendo en alto todo lo que se le ocurra, su proceso mental en la navegación y sus opiniones. De igual forma, al final de la prueba agradeceremos todos los comentarios que haga, tanto si son positivos como negativos.

9.7.3. Prototipo a evaluar

Para las sesiones de evaluación se siguió el prototipo de la interfaz incluido previamente a través de imágenes en esta memoria.

9.7.4. Cuestionario final

Tras la prueba del participante con la interfaz de la plataforma es recomendable realizarle un cuestionario para valorar su grado de satisfacción con esta para sacar conclusiones de cara a mejorarla. Siguiendo la idea de las preguntas que nos planteamos para la investigación realizamos un cuestionario Usefulness, Satisfaction, and Ease of use Questionnaire (USE) para medir la usabilidad subjetiva de nuestra plataforma. En este se le hicieron una serie de afirmaciones al usuario sobre la utilidad, facilidad de uso, facilidad de aprendizaje y satisfacción de nuestra plataforma que tiene que valorar del 1 al 7 siendo:

- 1- Totalmente en desacuerdo
- 2- En desacuerdo
- 3- Algo en desacuerdo
- 4- Ni de acuerdo ni en desacuerdo
- 5- Un poco de acuerdo
- 6- De acuerdo
- 7- Totalmente de acuerdo

Además, dejamos al participante que nos cuente su experiencia con sus propias palabras dejando un cuadro de texto tras cada una de las preguntas del cuestionario. Así, el usuario pudo relatar los problemas o dificultades que encontró al probar nuestra plataforma

El lector puede acceder al cuestionario final realizado accediendo al siguiente enlace. [2]
https://cutt.ly/jHbAM78.

[2] https://cutt.ly/jHbAM78

9.8. Desarrollo de las sesiones de evaluación

Las sesiones de evaluación con cada uno de los participantes elegidos se desarrollaron de la siguiente forma:

El miembro del equipo encargado de la evaluación estableció una sesión a una hora y día en el que ambas partes tuvieran disponibilidad para realizar una prueba de 10-15 minutos aproximadamente. Estas sesiones se llevaron a cabo todas presencialmente por comodidad y facilidad para la realización de la prueba.

Al inicio de la sesión, el participante rellenó el screening y se les presentó el proyecto desarrollado, siguiendo el guión. A continuación, los miembros del equipo encargado ejecutaron el proyecto en sus ordenadores personales, dejando libertad a los participantes para el manejo de la plataforma. Estos miembros les iban indicando las distintas tareas a realizar.

En el transcurso de la prueba, cada uno de los miembros prestó atención a las dificultades encontradas por los participantes. A su vez, se interesaba sobre el entendimiento de la plataforma y la posible aparición de problemas. Al final de la prueba se le pidió al participante hacer una evaluación de la plataforma indicando las funcionalidades que no había llegado a entender o no le habían gustado.

Por motivos de privacidad no grabaron las sesiones previamente descritas. Cada uno de los miembros del equipo que las realizaron se encargó de elaborar un informe que sintetizaba el desarrollo de la sesión, una vez esta había finalizado.

Tras finalizar la prueba, el miembro encargado del equipo facilitó al participante el enlace del cuestionario final para reflejar su experiencia, satisfacción y dar una valoración sobre nuestra plataforma, su utilidad y facilidad de uso y de aprendizaje.

Las descripciones individuales de las sesiones de evaluación se muestran en los anexos.

9.9. Resultados de la evaluación

Atendiendo a cada una de las preguntas realizadas en el cuestionario se sacaron los siguientes resultados.

El onboarding inicial tuvo una gran acogida entre todos los participantes. Les fue de gran ayuda para entender el funcionamiento de la página principal, destacando su sencillez e intuitividad. Por otro lado, uno de los entrevistados indicó que no le gustaban los círculos del primer slide del onboarding.

La página principal de la aplicación presentó más complicaciones respecto a la navegabilidad. A algunos les resultó complicado acceder a una empresa en concreto, pues para ello deben pulsar el botón 'Ver empresa' en el carousel que por su característica de desplazamiento hacía complicado pulsarlo, además de obligar al usuario a esperar a que la compañía en cuestión apareciera. Uno de los usuarios intentó acceder a la empresa desde la pantalla de información del proyecto, que no accede a la información de la empresa sino a la propia página web de esta. Otro trató de acceder desde la página Company desde el menú, encontrándose con la página en blanco muestra la información de las empresas al seleccionar una, pero no accediendo vía menú. Ninguno de ellos accedió clicando en el top 5. Para dar solución a tal problema se plantea añadir a la página Company la opción de

navegar entre las distintas empresas de forma más sencilla. A su vez, otro entrevistado no conocía el término ETH por lo que le causó confusión verlo y además indicó que aunque comprende las funcionalidades y contenidos le gustaría que hubiera menos módulos en un mismo lugar.

Las gráficas también fueron foco de división entre los participantes. Algunos no entendieron qué datos se mostraban, porque las leyendas eran escuetas. Sin embargo, otros declararon que sí eran sencillos de entender, a su vez interesante, pues reflejan datos que habitualmente no se dan a conocer en los medios de información más habituales y que además las empresas tratan de esconder. Para ellos las gráficas representan conceptos y cantidades fáciles de comprender e interpretar. En general, a excepción de los dos detalles mencionados anteriormente, los participantes vieron con buenos ojos la página principal, destacando su sencillez.

Uno de los usuarios no se acordaba del logo de LinkedIn, por lo que no fue capaz de hacer el login. Destacar que el resto indicó que este inicio de sesión les resultó más cómodo que el habitual inicio de sesión por Google, Facebook o creando una nueva cuenta.

Solo uno de los usuarios había usado previamente una cartera como Metamask, pero les permitimos trabajar con la cuenta del proyecto para evitarles crear una nueva y solicitar ETH para a Test. Ninguno encontró problemas en el proceso de pago para donar, que consiste solo en seleccionar un valor, una cuenta, conectarla y confirmar la donación. Sin embargo, indicaron no comprender la procedencia del coste que hay detrás de las denuncias que se indica en la tarjeta '¿Por qué donar?'.

Relativo al onboarding, todos destacan que es gracias a este que adquirieron rápidamente una idea general y mostraron especial afinidad con esta técnica. Confirman que si consideran la aplicación como intuitiva en cierta medida es por este recurso.

Para concluir, no mostraron inconveniente en incluir información personal en las denuncias y todos consideraron importante permitir la visualización de estas. Más aún, uno de los evaluados comentaba que a su parecer era uno de los atractivos de la plataforma el conocer las injusticias que se dan en cada una de las empresas, puesto que esa información rara vez escapa en entorno de la empresa. Los entrevistados también recalcaron la facilidad de navegación entre las distintas secciones de la aplicación.

9.10. Cambios tras prueba con usuarios

Tras la prueba con usuarios nos dimos cuenta de que existían algunos elementos de la plataforma que quedaban algo confusos. Por ello, tomamos nota de cada uno de ellos y procedimos a realizar los ajustes, que fueron los siguientes:

- Añadir en la pantalla de TFG Información el modo en que es calculada la reputación en nuestra plataforma. Los usuarios dejaron claro que no era transparente del todo la plataforma por no saber con qué criterio se calculaba la reputación de las distintas empresas.

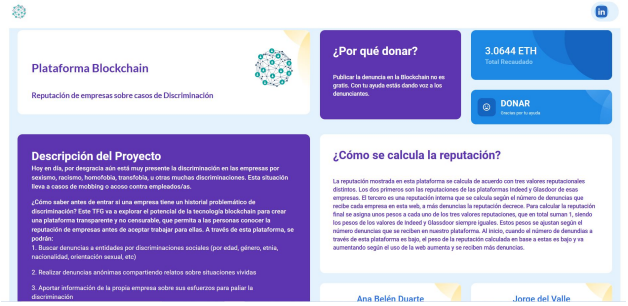


Figura 76: Diseño de TFG información
Fuente propia

- Cambiar el onboarding, ya que ahora, en lugar del índice es, posible acceder directamente a la información del proyecto desde la página de inicio. El resto de elementos del índice no eran necesarios, por lo que se decidió eliminar el índice, que era algo confuso, y acceder directamente a la pantalla de información del TFG.

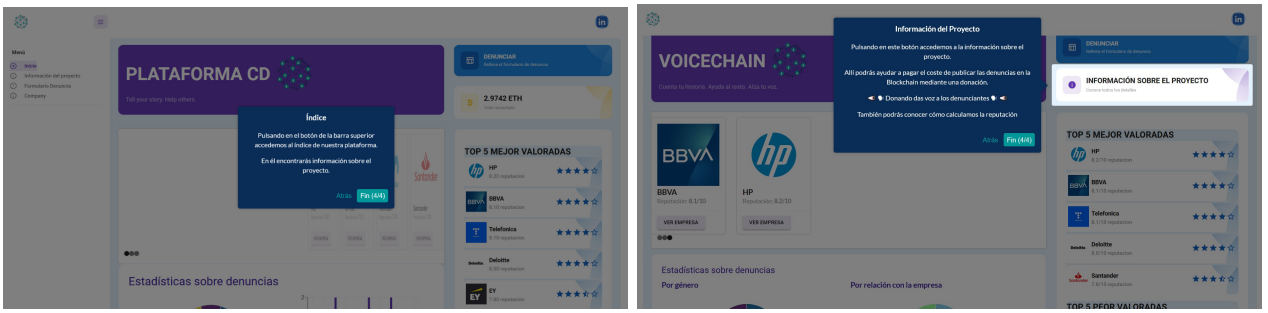


Figura 77: Diseño final del onboarding
Fuente propia

- Intercambiar el botón inicial de donar por uno de acceso a la pantalla de información. Esto se decidió puesto que los usuarios encontraban extraño que nada más entrar a la aplicación una de las cosas en las que más foco se ponía era en pedir donaciones y podrían ni si quiera aún saber de qué trataba el proyecto y para qué se utilizaría su donación. En su lugar, se decidió redirigir a los usuarios a toda la información sobre el proyecto, sus integrantes y objetivos.
- Además se decidió eliminar el menú puesto que ya se podía acceder a todas las páginas y funcionalidades de la plataforma desde otros botones.
- Finalmente, se introdujeron títulos a las gráficas de estadísticas sobre las denuncias impuestas ya que los usuarios no las entendían del todo únicamente con la leyenda y podían llevar a confusión.

10. Conclusiones y trabajo futuro

10.1. Límites del proyecto

Al ser el proyecto una DApp, muchos límites vienen impuestos por el concepto de descentralización y la tecnología **Blockchain**. A lo largo del desarrollo de la implementación, varias ideas han tenido que ser descartadas por estos límites. El principal es que todo lo que se almacena en blockchain es público, es decir, todo lo que se sube a la red es transparente. Esto es una gran ventaja cuando se busca una aplicación que pretenda transmitir confianza y transparencia, como es nuestro caso. Sin embargo, presenta el problema de la privacidad. Accediendo a la web *Etherscan* [163] y conociendo la dirección del contrato principal, cualquier persona puede ver todos los hashes de las denuncias. Como IPFS también es un sistema descentralizado público, introduciendo el correspondiente hash en [164], se obtienen todos los datos introducidos por el denunciante.

La capa del sistema centralizado ha conseguido que obtener el nombre y apellidos del denunciante dependa de un ataque a LinkedIn. Esto es un punto muy fuerte del proyecto, aunque pudieran darse situaciones en las que personas del entorno del denunciante pudieran conocer su identidad. Es importante resaltar que son los denunciantes los que deciden que campos rellenan en la denuncia (tan solo tipo y descripción del suceso son obligatorios) y son informados del carácter público de los datos emitidos en la denuncia. Aunque el uso de LinkedIn evita almacenar datos personales en el sistema centralizado, crea al usuario la necesidad de confiar en una entidad de terceros; siendo esta una situación que no se produciría si se pudiera utilizar un sistema que permitiera la identificación de forma descentralizada.

Otra notable limitación de blockchain, y más en concreto de Ethereum, es el coste que supone ejecutar cualquier transacción o llamada a una función de un contrato inteligente. Lo ideal sería que nadie tuviera que pagar nada por emitir una denuncia o visualizar cuántas ha recibido una empresa. Para evitar que las víctimas deban financiar su propia denuncia, incentivando el hecho de denunciar, que ya de por si es duro, los creadores del proyecto han creado una cuenta en Ethereum que costea todos los gastos. A través de donaciones, todo aquel que lo desee puede contribuir. Por lo tanto, si en algún momento la cuenta se queda sin ethers, los usuarios de la aplicación perderán el acceso a la funcionalidad principal de la aplicación.

En adición, se deben tener en cuenta las limitaciones del proyecto derivadas del empleo de herramientas en su versión *community* y la falta de componentes hardware. La base de datos no posee *replicación*, por lo que en caso de que se perdiera la información debido a un error, no habría otros nodos que pudieran aportarla. Tampoco se pueden ejecutar los servidores en distintas máquinas ni sacar el máximo potencial de los servicios de cloud. Al ser ejecutado el Back-End de forma local, tampoco es posible escalar la aplicación lo suficiente como para tratar una cantidad extremadamente alta de peticiones ya que el equipo no se posee el poder el poder computacional suficiente independientemente de lo ligeros que sean los contenedores de Docker.

Por último, el tiempo que toma la realización de cada transacción afecta a la experiencia del usuario en la aplicación ya que este tiene que esperar a que esta se acabe de confirmar, lo cuál puede variar en gran medida. Las transacciones realizadas también suponen un coste energético e impacto medioambiental muy elevados como se puede observar en [165]. No obstante, con la llegada de *Ethereum 2.0* esta limitación se reducirá considerablemente ya que está siendo desarrollado con la eficiencia energética como uno de sus objetivos.

10.2. Trabajo Futuro

Como comentamos al comienzo del documento, en el miró clasificamos los post-it por colores, asignando el color amarillo para aquellas ideas o mejoras que dejamos como trabajo futuro. Entre ellas se encuentran las siguientes:

- Crear una especie de comunidad de denunciantes, donde las personas con casos similares puedan interactuar y comunicarse por medio de un chat en la propia aplicación o mediante el correo electrónico.
- Mejorar la información relativa a cada empresa, pudiendo denunciar por departamento o ciudad, pues, como sabemos, el trato de cada empresa en cada región y departamento puede diferir.
- Dividir el cuestionario en dos secciones diferentes, que separe las preguntas más generales de las más específicas, relacionadas con la denuncia.

Asimismo, podríamos tratar de añadir nuevas funcionalidades a la plataforma, a parte de denunciar, como por ejemplo conocer cómo es el trato en algún departamento de una empresa, para ayudar a futuros trabajadores en trámites de incorporación o cómo de fácil es ascender dentro de la empresa o subir de sueldo.

Estos cambios se traducen en que el contrato principal *ReputationControl* debe almacenar más cantidad de información, como por ejemplo hashes de comentarios asociados a cada denuncia. En función de los cambios elegidos, se debería crear un nuevo contrato. Por otra parte, en cuanto a los tokens, sería interesante proponer un nuevo estándar de token reputacional, no transferible, a la comunidad de Ethereum. Actualmente todos los EIPs de este estilo se encuentran en un estado de Draft, y su uso está desaconsejado. Por lo tanto, parece una buena oportunidad tratar de desarrollar un ERC que se ajuste a los requisitos de los tokens reputacionales, tanto negativos como positivos. Además, esto generaría un gran impacto en la comunidad, aportando repercusión a la aplicación. Los pasos a seguir para proponer una EIP en el github de Ethereum vienen explicados aquí. [166]

En función del reclamo de la aplicación, otra posibilidad respecto a los tokens sería darle más visualización en la interfaz. En la aplicación no se distingue cuántos tokens posee cada empresa, pues estos se agregan a una fórmula general. Podrían mostrarse por separado. De hecho, si la idea de los tokens reputacionales negativos resulta confusa, se cambiaría a tokens positivos, es decir, cuántos más tenga una empresa mejor. Este caso es similar al descrito en el apartado de Implementación de Tokens, en el que se propone un sistema de tokens transferibles entre usuarios, como si fuera un mercado. Las empresas no podrían transferirlos, pero para denunciar los usuarios deben comprar tokens y gastarlos en las propias denuncias.

Otra de las modificaciones que se podrían añadir en el futuro en el sistema descentralizado es la herramienta zk-SNARK [167], pues podría aportar veracidad a las denuncias sin necesidad de revelar la identidad del usuario. Esta tecnología está basada en el concepto de conocimiento cero, por lo que encajaría perfectamente en el proyecto.

Por otra parte, la aplicación web podría desplegarse mediante un servicio de hosting de forma que pudiera ser accedida y empleada públicamente. El servidor a su vez se incluiría en un clúster de Kubernetes [168], un conjunto de nodos o máquinas que ejecutan aplicaciones en contenedores. De esta forma se podría garantizar una mayor disponibilidad y se aprovecharía al máximo el potencial

de los contenedores de Docker mediante la ejecución de distintas réplicas del servidor en los nodos disponibles.

Otra labor a realizar es la mejora del rendimiento de la aplicación. Mediante herramientas de *profiling* como el *plugin de Devtools* se puede observar si se producen renderizados innecesarios, el consumo de CPU y detectar los componentes que se comportan como un cuello de botella en la aplicación, reduciendo el rendimiento global de esta.

De cara al futuro sería también necesaria la realización de *análisis de riesgos y vulnerabilidades*, así como el desarrollo de un plan de mitigación de riesgos con las contramedidas a adoptar en caso de que estos se materialicen. Ya que tratamos con denuncias realizadas por usuarios es de vital importancia reducir la probabilidad de que ocurra un incidente de seguridad, por lo que sería ideal revisar y actualizar el código de la aplicación periódicamente para poder crear una plataforma robusta basada en la privacidad desde el diseño. Gracias a plataformas como *CVE* [169] y *OWASP* [170] y la página oficial del INCIBE-CERT [171], uno de los centros de respuesta a incidentes del Instituto Nacional de Ciberseguridad, se podrían buscar las vulnerabilidades presentes y determinar si alguna de las herramientas o librerías ha quedado obsoleta de forma que se actualice a una nueva versión más estable o se cambie por otra más segura.

Finalmente, se podrían realizar evaluaciones acerca de la accesibilidad de la aplicación basadas en los *principios de accesibilidad web de WCAG* [172] de forma que la aplicación pueda adaptarse para ser utilizada por la mayor cantidad posible de usuarios de forma cómoda y sencilla. Ejemplos de cambios para acomodar la aplicación a las distintas necesidades de los usuarios serían añadir la posibilidad de cambiar el tamaño del texto y contraste de colores, permitir la navegación mediante teclado y utilizar descripciones claras en las distintas imágenes y contenido interactivo.

10.3. Conclusión

Para poder llevar a cabo este proyecto con el mejor resultado posible, todos los miembros del grupo han adquirido grandes conocimientos en diversas ramas tecnológicas, destacando Blockchain, Node.js, Docker y React. Cada una de ellas tiene un papel esencial en el desarrollo del TFG, y todas juntas proporcionan un potente núcleo a partir del cual crear otras aplicaciones. React es una de las principales herramientas para desarrollar aplicaciones web, por lo que conocer su funcionamiento es muy beneficioso. A su vez, Node.js posee una gran relevancia como entorno para Back-End, gracias a su extensa comunidad que contribuye a su expansión y funcionalidad mediante la constante creación de frameworks y módulos.

Por otra parte, Docker permite la adaptación de la aplicación a cualquier entorno así como dar cabida a un escalado sencillo sin consumir importantes cantidades de recursos. Es una tecnología muy útil debido al elevado número de imágenes disponibles en Docker Hub y otros repositorios, sobre las cuales basarse para hacer distintos tipos de aplicaciones o microservicios.

En cuanto a Blockchain, específicamente Ethereum, es la piedra angular del proyecto, que permite diferenciar esta aplicación de toda la competencia. Su carácter público y descentralizado ofrece a las víctimas la seguridad de que no serán ignoradas ni silenciadas. Todos los miembros se han especializado en esta rama y son capaces de desarrollar contratos inteligentes, robustos y eficientes.

En adición, se adquirieron conocimientos sobre el uso de plantillas para la creación y la utili-

zación de frameworks para frond-end, que permiten el desarrollo de una interfaz gráfica más rica, intuitiva y completa. Los componentes editables definidos en estos frameworks, favorecen la customización de la aplicación; pudiendo construir así la identidad visual de la plataforma.

El proceso de desarrollo también ha conllevado en gran medida al aprendizaje del modelo de programación asíncrona, cuya importancia y uso está creciendo exponencialmente. Este estilo de programación ha permitido que en nuestra aplicación se puedan ejecutar varios procesos a la vez siempre y cuando no sean dependientes entre sí, resultando en una aceleración considerable en comparación con el empleo de un modelo clásico.

Mediante este proyecto ha sido posible demostrar la sinergia y capacidad de colaboración que puede existir entre los sistemas centralizados y descentralizados, combinando las arquitecturas con la finalidad de extraer el máximo potencial de sus respectivas ventajas. El sistema descentralizado ha resultado crucial a la hora de aportar persistencia y transparencia a las denuncias y reputaciones internas, mientras que el sistema centralizado ha permitido un almacenamiento y devolución de datos eficiente y veloz sin coste monetario por transferencia. Adicionalmente, se ha conseguido implementar una forma de autenticación que permita identificar al usuario con el perfil buscado sin que este tenga que ofrecer su privacidad a cambio, aportando de esta forma el anonimato deseado para situaciones tan delicadas como lo es la realización de una denuncia.

Asimismo, ha contribuido a reflejar que la tecnología Blockchain es perfectamente apta para su uso en aplicaciones web dedicadas a una gran multitud de temáticas; superando el uso de herramientas orientadas a sistemas centralizados en ciertos ámbitos. Algunos ejemplos de uso efectivo podrían ser su inclusión en cadenas de suministro y sistemas de votaciones para juntas de accionistas. Debido a la temática de este trabajo y la eficacia observada a la hora de desempeñar las distintas funcionalidades designadas, cabe destacar su adecuación a la hora de emplearse en proyectos destinados a fines de índole social. Dada su naturaleza descentralizada, impide que las distintas entidades puedan influir en los datos mostrados en el sistema, otorgando por ende el poder a sus usuarios.

Por consiguiente, consideramos que se ha realizado de forma exitosa la creación de una aplicación web de carácter social que incorpore la tecnología Blockchain como principal componente. Se han logrado cumplir los objetivos definidos, aportando las funcionalidades que se han considerado necesarias a la aplicación. Además, la implementación resultante, permite la integración de nuevas funcionalidades de cara a futuras actualizaciones que ayuden a reducir la vulneración de los derechos de los trabajadores.

11. Conclusions and future work

11.1. Limits of the project

As the project is a DApp, many limits are imposed by the concept of decentralization and the **Blockchain** technology. Throughout the development of the implementation, several ideas have had to be ruled out by these limits. The main one is that everything that is stored in the Blockchain is public, which implies that everything that is uploaded to the network is transparent. This is a great advantage when looking for an application that aims to convey trust and transparency, as is our case. However, it presents the problem of privacy. Accessing the web *Etherscan* [163] and knowing the address of the main contract, anyone can see all the hashes of the complaints. Since IPFS is also a public decentralized system, by entering the corresponding hash in [164], all the data entered by the whistleblower is obtained.

The centralized system layer has made obtaining the users's first and last name dependent on an attack on LinkedIn. This is a very strong point of the project, although there could be situations in which people from the complainant's environment could know their identity. It is important to highlight that it is the users who decide which fields to fill in the complaint (only the type and description of the event are required) and are informed of the public nature of the data issued in the complaint. Although the use of LinkedIn avoids storing personal data in the centralized system, it creates the need for the user to trust a third party entity; this being a situation that would not occur if a system could be used that would allow decentralized identification.

Another notable limitation of blockchain, and more specifically of Ethereum, is the cost of executing any transaction or call to a smart contract function. Ideally, no one would have to pay anything to issue a complaint or see how many a company has received. To prevent victims from having to finance their own complaint, encouraging the fact of reporting, which is already hard, the members of the project have created an account in Ethereum that pays for all expenses. Through donations, anyone who wants to can contribute. Therefore, if at any time the account runs out of ether, the app users will lose access to the main functionality of the app.

In addition, the limitations of the project derived from the use of tools in its *community* version and the lack of hardware components must be taken into account. The database does not have *replication*, so in case the information is lost due to an error, there would be no other nodes that could provide it. Nor can the servers be run on different machines or take full advantage of the cloud services. As the Back-End is executed locally, it is also not possible to scale the application enough to deal with an extremely high number of requests since the team does not have enough computational power regardless of how light the containers are. Docker.

Lastly, the time it takes to complete each transaction affects the user's experience in the application since they have to wait for the transaction to finish confirming, which can vary greatly. The transactions carried out also entail a very high energy cost and environmental impact, as can be seen in [165]. However, with the arrival of *Ethereum 2.0* this limitation will be considerably reduced since it is being developed with energy efficiency as one of its objectives.

11.2. Future Work

As we commented at the beginning of the document, in it we classify the post-its by colors, assigning the yellow color to those ideas or improvements that we leave as future work. Among

them are the following:

- Create a kind of community of claimants, where people with similar cases can interact and communicate through a chat in the application itself or by email.
- Improve the information related to each company, being able to report by department or city, since, as we know, the treatment of each company in each region and department may differ.
- Divide the questionnaire into two different sections, separating the more general questions from the more specific ones, related to the complaint.

Likewise, we could try to add new functionalities to the platform, apart from reporting, such as knowing how the treatment is in some department of a company, to help future workers in incorporation procedures or how easy it is to ascend within the company or salary increase.

These changes mean that the main *ReputationControl* contract must store more information, such as comment hashes associated with each report. Depending on the chosen changes, a new contract should be created. On the other hand, regarding tokens, it would be interesting to propose a new, non-transferable, reputational token standard to the Ethereum community. Currently all EIPs of this style are in a Draft state, and their use is discouraged. Therefore, it seems like a good opportunity to try to develop an ERC that fits the requirements of reputational tokens, both negative and positive. In addition, this would generate a great impact on the community, bringing repercussion to the application. The steps to follow to propose an EIP on the Ethereum github are explained here. [166]

Based on the claim of the application, another possibility regarding the tokens would be to give it more visualization in the interface. The application does not distinguish how many tokens each company has, since these are added to a general formula. They could be shown separately. In fact, if the idea of negative reputational tokens is confusing, it would be changed to positive tokens, that is, the more a company has, the better. This case is similar to the one described in the Token Implementation section, in which a system of transferable tokens between users is proposed, as if it were a market. Companies could not transfer them, but to report users must buy tokens and spend them on the reports themselves.

Another of the modifications that could be added in the future to the decentralized system is the zk-SNARK [167] tool, since it could provide veracity to the complaints without the need to reveal the user's identity. This technology is based on the concept of zero knowledge, so it would fit perfectly into the project.

On the other hand, the web application could be deployed through a hosting service in such a way that it could be accessed and used publicly. The server in turn would be included in a Kubernetes cluster [168], a set of nodes or machines running containerized applications. This would ensure higher availability and take full advantage of the potential of Docker containers by running different replicas of the server on available nodes.

Another task to be done is to improve the performance of the application. Using *profiling* tools such as the *Devtools plugin* you can see if unnecessary renderings occur, CPU consumption and detect components that behave as a bottleneck in the application, reducing overall performance this.

Looking to the future, it would also be necessary to carry out *risk and vulnerability analysis*, as well as the development of a risk mitigation plan with the countermeasures to be adopted in the event that these materialize. Since we deal with complaints made by users, it is of vital importance to reduce the probability of a security incident, so it would be ideal to periodically review and update the application code in order to create a robust platform based on privacy by design. . Thanks to platforms such as *CVE* [169] and *OWASP* [170] and the official website of INCIBE-CERT [171], one of the incident response centers of the National Institute of Cybersecurity, the present vulnerabilities could be searched for and determine if any of the tools or libraries has become obsolete so that it can be updated to a new, more stable version or changed to a more secure one.

Finally, evaluations could be made about the accessibility of the application based on the *WCAG web accessibility principles* [**accessibility**] so that the application can be adapted to be used by the greatest possible number of users in a consistent way: comfortable and simple. Examples of changes to accommodate the application to the different needs of users would be adding the possibility to change the text size and color contrast, allowing keyboard navigation and using clear descriptions in the different images and interactive content.

11.3. Conclusion

In order to carry out this project with the best possible result, all the members of the group have acquired great knowledge in various technological branches, highlighting Blockchain, Node.js, Docker and React. Each one of them has an essential role in the development of the Final Degree Project, and together they provide a powerful core from which to create other applications. React is one of the main tools for developing web applications, thus knowing how it works is very beneficial. Moreover, Node.js is highly relevant as a Back-End environment, thanks to its extensive community that contributes to its expansion and functionality through the constant creation of frameworks and modules

On the other hand, Docker allows the adaptation of the application to any environment as well as allowing for easy scaling without consuming significant amounts of resources. It is a very useful technology due to the high number of images available in Docker Hub and other repositories, on which to build different types of applications or microservices.

As for Blockchain, specifically Ethereum, it is the cornerstone of the project, which allows this application to be differentiated from all the competition. Its public and decentralized nature offers victims the security that they will not be ignored or silenced. All members have specialized in this branch and are capable of developing smart, robust and efficient contracts.

In addition, knowledge was acquired on the use of templates for the creation and use of frameworks for frond-end, which allow the development of a richer, more intuitive and complete graphical interface. The editable components defined in these frameworks favor the customization of the application; thus being able to build the visual identity of the platform.

The development process has also largely led to the learning of the asynchronous programming model, whose importance and use is growing exponentially. This programming style has allowed our application to execute several processes at the same time as long as they are not dependent on each other, resulting in a considerable acceleration compared to the use of a classic model.

Through this project it has been possible to demonstrate the synergy and collaboration capacity that can exist between centralized and decentralized systems, combining the architectures in order to extract the maximum potential of their respective advantages. The decentralized system has been crucial in bringing persistence and transparency to reports and internal reputations, while the centralized system has enabled fast and efficient storage and return of data with no monetary cost per transfer. Additionally, it has been possible to implement a way of authentication that allows the user to be identified with the profile sought without the latter having to offer their privacy in exchange, thus providing the desired anonymity for situations as delicate as filing a complaint.

Likewise, it has contributed to reflecting that Blockchain technology is perfectly suitable for use in web applications dedicated to a large number of topics; overcoming the use of tools oriented towards centralized systems in certain areas. Some examples of effective use could be its inclusion in supply chains and voting systems for shareholder meetings. Due to the theme of this work and the effectiveness observed when performing the different functions designated, it is worth highlighting its suitability when used in projects for social purposes. Given its decentralized nature, it prevents the different entities from influencing the data displayed in the system, thus granting power to its users.

Therefore, we consider that the creation of a social web application that incorporates Blockchain technology as the main component has been successfully carried out. The defined objectives have been achieved, providing the functionalities that have been considered necessary to the application. In addition, the resulting implementation allows the integration of new functionalities for future updates that help reduce the violation of workers' rights.

Bibliografía

[1] Vitalik Buterin. *The Meaning of Decentralization*. 2017 [En línea] Disponible en: https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274 [Accedido: ene-2022].

[2] Vitalik Buterin. *Ehtereum WhitePaper*. [En línea] Disponible en: https://ethereum.org/en/whitepaper/#introduction-to-bitcoin-and-existing-concepts [Accedido: oct-2021].

[3] FT. *Cómo funciona blockchain*. 2017 [En línea] Disponible en: https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda [Accedido: ene-2021].

[4] Yining Hu y col. "The Use of Smart Contracts and Challenges". En: *CoRR* abs/1810.04699 (2018). arXiv: 1810.04699. URL: http://arxiv.org/abs/1810.04699.

[5] Laura A.Linn y Martha B.Koo. "Blockchain for health data and its potential use in health it and health care related research". En: (2019. [En línea] Disponible en: https://www.healthit.gov/sites/default/files/11-74-ablockchainforhealthcare.pdf [Accedido: may-2022]).

[6] uPort. *Open identity system for the decentralized web.* 2017 [En línea] Disponible en: https://www.uport [Accedido: may-2021].

[7] Sovrin. *Sovrin: Control your digital identity*. 2019 [En línea] Disponible en: https://sovrin.org [Accedido: may-2021].

[8] Watson. *Watson internet of things*. 2019 [En línea] Disponible en: https://www.ibm.com/internet-of-things/trending/blockchain [Accedido: may-2021].

[9] Chain of things. *Chain of things*. 2019 [En línea] Disponible en: https://www.chainofthings.com [Accedido: may-2021].

[10] *Stellar*. 2019 [En línea] Disponible en: https://www.stellar.org [Accedido: may-2021].

[11] *Hyperledger*. 2019 [En línea] Disponible en: https://www.hyperledger.org [Accedido: may-2021].

[12] Saveen A. Abeyratne, Radmehr P. Monfared. "BLOCKCHAIN READY MANUFACTURING SUPPLY CHAIN USING DISTRIBUTED LEDGER". En: *IJRET: International Journal of Research in Engineering and Technology* (2016.[En línea] Disponible en: https://ijret.org/volumes/2016v05/i09/IJRET20160509001.pdf[Accedido: may-2022]).

[13] *Danish political party may be first to use block chain for internal voting*. 2014 [En línea] Disponible en: http://www.newsbtc.com/2014/04/22/danish-political-party-may-first-use-block-chain-internal-voting [Accedido: may-2021].

[14] P. McCorry, S. F. Shahandashti, and F. Hao. "A smart contract forboardroom voting with maximum voter privacy." En: *Financial Cryptography and Data Security.* (2016.[En línea] Disponible en: https://doi.org/10.1007/978-3-319-70972-7_20[Accedido: may-2022]).

[15] B. Cant, A. Khadikar, A. Ruiter, J. Bronebakk, J. Coumaros, J. Buvat, and A. Gupta. "Smart contracts in financial services: Getting from hype to reality." En: *Capgemini Consulting* (2016.[Accedido: may-2022]).

[16] Dan Zaitsev. *Top Blockchains to Develop DApps in 2022*. [En línea] Disponible en: https://medium.com/@dan_zaitsev/top-blockchains-to-develop-dapps-in-2022-21c5b30635f1 [Accedido: dic-2021].

[17] Hassan, S. De Filippi, P. "Decentralized Autonomous Organization". En: *Internet Policy Review* 10(2) (2021. [En línea] Disponible en: https://doi.org/10.14763/2021.2.1556 [Accedido: may-2022]).

[18] Luis Cuende. *A DAO is an internet-native entity with no central management which is regulated by a set of automatically enforceable rules on a public blockchain, and whose goal is to take a life of its own and incentivize people to achieve a shared common mission.* [Tuit] 2020 [En línea] Disponible en: https://twitter.com/licuende/status/1263511552709267456? [Accedido: may-2022].

[19] A. Rea, D. Kronovet, A. Fischer y J. du Rose. *COLONY Technical White Paper*. 2020 [En línea] Disponible en: https://colony.io/whitepaper.pdf [Accedido: ene-2021].

[20] Protocol Labs. *IPFS powers the Distributed Web*. [En línea] Disponible en: https://ipfs.io/ [Accedido: abr-2022].

[21] Stefania Milan. "Data activism as the new frontier of media activism". En: *Media Activism in the Digital Age* (jul. de 2017 [En línea] Disponible en: https://www.taylorfrancis.com/chapters/edit/10.4324/9781315393940-13/[Accedido: may-2022]), pág. 13. DOI: 10.4324/9781315393940-13.

[22] Ishwarlal Hingorani y col. "Police Complaint Management System using Blockchain Technology". En: *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*. 2020, págs. 1214-1219. DOI: 10.1109/ICISS49785.2020.9315884.

[23] Mijanur Rahman, Moshiul Azam y Faria Sanjida Chowdhury. "Secure Complaint Management System against Women Harassment at Workplace Using Blockchain Technology". En: *International Journal of Electrical and Computer Engineering Systems (IJECES)* 13 (2022. [En línea] Disponible en: https://ijeces.ferit.hr/index.php/ijeces/article/view/661/161 [Accedido: feb-2022]).

[24] Audun Jøsang, Roslan Ismail y Colin Boyd. "A survey of trust and reputation systems for online service provision". En: *Decision Support Systems* 43.2 (2007), págs. 618-644. DOI: https://doi.org/10.1016/j.dss.2005.05.019. URL: https://www.sciencedirect.com/science/article/pii/S0167923605000849.

[25] Daniel Tedesco. *EIP-4974: Experience (EXP) Token Standard [DRAFT]*. [En línea] Disponible en: https://eips.ethereum.org/EIPS/eip-4974 [Accedido: feb-2022].

[26] Ethereum. *ERC20*. [En línea] Disponible en: https://ethereum.org/es/developers/docs/standards/tokens/erc-20/ [Accedido: feb-2022].

[27] Ismail, Roslan and Josang, Audun. "The Beta Reputation System". En: *BLED 2002 Proceedings. 41.* (2002. [En línea] Disponible en: https://aisel.aisnet.org/bled2002/41 [Accedido: abr-2022]).

[28] Ahmed S. Almasoud. "Smart contracts for blockchain-based reputation systems". En: *University of Technology Sydney* (2020 [En línea] Disponible en: https://opus.lib.uts.edu.au/handle/10453/143883 [Accedido: may-2022]).

[29] Ethereum Foundation. *Solidity logo*. 2015 [En línea] Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Solidity_logo.svg [Accedido: may-2022].

[30] Develop Paper. *From bitcoin script engine to Ethereum virtual machine*. 2020 [En línea] Disponible en: https://developpaper.com/from-bitcoin-script-engine-to-ethereum-virtual-machine/ [Accedido: may-2022].

[31] Ethereum Foundation. *Ethereum logo 2014*. 2014 [En línea] Disponible en: https://commons.wikimedia.org/wiki/File:Ethereum_logo_2014.svg [Accedido: may-2022].

[32] alchemy. *Obtain free ethers for Rinkeby.* [En línea] Disponible en: https://rinkebyfaucet.com/ [Accedido: dic-2021].

[33] OpenZeppelin. *OpenZeppeling Logo.* [En línea] Disponible en: https://www.openzeppelin.com/ [Accedido: may-2022].

[34] Remix IDE. *Remix Logo.* [En línea] Disponible en: https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f [Accedido: may-2022].

[35] Infura Inc. *Infura Logo.* [En línea] Disponible en: https://blog.infura.io/post/infura-launches-support-for-optimistic-ethereum [Accedido: may-2022].

[36] Web3 Labs. *Web3j Logo.* [En línea] Disponible en: https://docs.web3j.io/4.8.7/ [Accedido: may-2022].

[37] Ethers.js. *Ethers.js logo.* [En línea] Disponible en: https://medium.com/@al5031/new-course-intro-to-ethers-js-dd2455387e69 [Accedido: may-2022].

[38] MetaMask. *MetaMask Fox.* 2018 [En línea] Disponible en: https://commons.wikimedia.org/wiki/File:MetaMask_Fox.svg [Accedido: may-2022].

[39] Protocol Labs. *IPFS logo.* [En línea] Disponible en: https://protocol.ai/work/ [Accedido: may-2022].

[40] strongDM. *MongoDB Atlas IAM.* [En línea] Disponible en: https://www.strongdm.com/loves/mongodb-atlas [Accedido: may-2022].

[41] MongoDB Inc. *Base de datos.* [En línea] Disponible en: https://www.mongodb.com/es/atlas/database [Accedido: dic-2021].

[42] Node.js. *Node.js Logo.* 2016 [En línea] Disponible en: https://nodejs.org/about/resources/ [Accedido: may-2022].

[43] Node.js. *Acerca de Node.js.* [En línea] Disponible en: https://nodejs.org/es/about/ [Accedido: nov-2021].

[44] merencia. *Node Cron.* [En línea] Disponible en: https://www.npmjs.com/package/node-cron [Accedido: abr-2022].

[45] extrabacon, brucedjones y almenon. *python-shell.* [En línea] Disponible en: https://www.npmjs.com/package/python-shell [Accedido: abr-2022].

[46] Nginx. *NGINX-product-icon.* [En línea] Disponible en: https://www.nginx.com/ [Accedido: may-2022].

[47] Express. *Express.* 2017 [En línea] Disponible en: https://expressjs.com/ [Accedido: nov-2021].

[48] Mongoose.js. [En línea] Disponible en: https://twitter.com/mongoosejs [Accedido: may-2022].

[49] Passport.js. [En línea] Disponible en: https://twitter.com/passportjs [Accedido: may-2022].

[50] LinkedIn. *LinkedIn logo initials.* 2014 [En línea] Disponible en: https://developer.linkedin.com/documents/branding-guidelines [Accedido: may-2022].

[51] SmartBear. *swagger_logo.* [En línea] Disponible en: https://swagger.io/tools/swagger-ui/ [Accedido: may-2022].

[52] Cameron Oelsen. *Jupyter Logo.* 2016 [En línea] Disponible en: https://github.com/jupyter/jupyter.github.io/blob/master/assets/main-logo.svg [Accedido: may-2022].

[53] codemaster138. *Axios logo*. 2019 [En línea] Disponible en: https://github.com/axios/axios-docs/blob/908d04c524e088ae7fde8a57a527e54710a4a5ab/assets/logo.svg [Accedido: may-2022].

[54] Axios. *Getting Started*. [En línea] Disponible en: https://axios-http.com/docs/intro [Accedido: dec-2021].

[55] Facebook. *React-icon*. [En línea] Disponible en: https://raw.githubusercontent.com/reactjs/reactjs.org/main/src/icons/logo.svg [Accedido: may-2022].

[56] Google. *Google Sheets Logo*. [En línea] Disponible en: https://workspace.google.com/intl/es-419/products/sheets/ [Accedido: may-2022].

[57] Material UI. *MUI Logo*. [En línea] Disponible en: https://mui.com/ [Accedido: may-2022].

[58] Material UI. *Move faster with intuitive React UI tools*. [En línea] Disponible en: https://mui.com/ [Accedido: abr-2022].

[59] CodedThemes. *Build Your Next Project WithBerry*. [En línea] Disponible en: https://berrydashboard.io/ [Accedido: abr-2022].

[60] Semantic UI. *React Semantic UI Logo*. [En línea] Disponible en: https://react.semantic-ui.com/ [Accedido: may-2022].

[61] GitHub. *Octicons-mark-github*. 2018 [En línea] Disponible en: https://github.com/github/octicons [Accedido: may-2022].

[62] Microsoft. *Visual Studio Code 1.35 icon*. 2019 [En línea] Disponible en: https://code.visualstudio.com/brand [Accedido: may-2022].

[63] Alberto Díaz. *docker-logo*. [En línea] Disponible en: https://blogs.encamina.com/por-una-nube-sostenible/contenedores-en-azure-devops-con-docker/docker-logo/ [Accedido: abr-2022].

[64] Docker. *Use containers to Build, Share and Run your applications*. [En línea] Disponible en: https://www.docker.com/resources/what-container/ [Accedido: abr-2022].

[65] Weaveworks. *Docker vs Virtual Machines (VMs) : A Practical Guide to Docker Containers and VMs*. [En línea] Disponible en: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms [Accedido: abr-2022].

[66] Docker. *Overview of Docker Compose*. [En línea] Disponible en: https://docs.docker.com/compose/ [Accedido: abr-2022].

[67] Microsoft Corporation. *Microsoft Office Teams*. 2019 [En línea] Disponible en: https://commons.wikimedia.org/wiki/File:Microsoft_Office_Teams_(2018%E2%80%93present).svg [Accedido: may-2022].

[68] Google. *Google Meet icon*. 2020 [En línea] Disponible en: https://commons.wikimedia.org/wiki/File:Google_Meet_icon_(2020).svg [Accedido: may-2022].

[69] Google. *Google Drive logo*. 2015 [En línea] Disponible en: https://commons.wikimedia.org/wiki/File:Google_Drive_logo.png [Accedido: may-2022].

[70] Fabián Alexis. *Antu trello*. 2016 [En línea] Disponible en: https://upload.wikimedia.org/wikipedia/commons/1/17/Antu_trello.svg [Accedido: may-2022].

[71] Miró. *Miró Logo*. [En línea] Disponible en: https://miro.com [Accedido: may-2022].

[72] Figma. *Figma Logo*. 2019 [En línea] Disponible en: figma.com [Accedido: may-2022].

[73] MDN Contributors. *MVC*. 2020 [En línea] Disponible en: https://developer.mozilla.org/es/docs/Glossary/MVC [Accedido: ene-2022].

[74] Ethereum.org. *ERC*. [En línea] Disponible en: https://ethereum.org/es/developers/docs/standards/tokens/erc-20/ [Accedido: ene-2022].

[75] Profesional Review. *Metamask*. 2021 [En línea] Disponible en: https://www.profesionalreview.com/2021/11/02/metamask-la-mejor-wallet-para-almacenar-tus-ethereum/ [Accedido: oct-2021].

[76] Web3 Labs Ltd. *Web3j*. [En línea] Disponible en: https://docs.web3j.io/4.8.7/ [Accedido: dic-2021].

[77] Ethers. *Ethers*. [En línea] Disponible en: https://docs.ethers.io/v5/ [Accedido: dic-2021].

[78] Ethers. *Provider*. [En línea] Disponible en: https://docs.ethers.io/v5/api/providers [Accedido: dic-2021].

[79] Ethereum. *Nodos en Ethereum*. [En línea] Disponible en: https://ethereum.org/es/developers/docs/nodes-and-clients/ [Accedido: dic-2021].

[80] Infura. *Infura*. [En línea] Disponible en: https://docs.infura.io/infura [Accedido: dic-2021].

[81] QuickNode. *ABI*. [En línea] Disponible en: https://www.quicknode.com/guides/solidity/what-is-an-abi [Accedido: nov-2021].

[82] SkillPlus. *ABI Image*. [En línea] Disponible en: https://skillplus.web.id/smart-contract-abi/ [Accedido: may-2022].

[83] IPFS. *API IPFS*. [En línea] Disponible en: https://docs.ipfs.io/reference/http/api/ [Accedido: may-2022].

[84] Express. *Routing*. 2017 [En línea] Disponible en: https://expressjs.com/en/guide/routing.html [Accedido: nov-2021].

[85] Express. *Community*. 2017 [En línea] Disponible en: https://expressjs.com/en/resources/community.html [Accedido: may-2021].

[86] Mongoose. *Schemas*. 2021 [En línea] Disponible en: https://mongoosejs.com/docs/guide.html [Accedido: nov-2021].

[87] Mongoose. *Models*. 2021 [En línea] Disponible en: https://mongoosejs.com/docs/models.html [Accedido: nov-2021].

[88] MongoDB Inc. *MongoDB Driver*. 2022 [En línea] Disponible en: https://mongodb.github.io/node-mongodb-native/ [Accedido: abr-2022].

[89] Bo Zhao. "Web scraping". En: *Encyclopedia of big data* (2017 [En línea] Disponible en: https://cutt.ly/vHykGqV [Accedido: abr-2022]), págs. 1-3.

[90] Passport.js. *Overview*. [En línea] Disponible en: https://www.passportjs.org/concepts/authentication/ [Accedido: nov-2021].

[91] U. Pavlovic y G. Atkins. *Docker: Top 7 Benefits of Containerization*. 2020 [En línea] Disponible en: https://hentsu.com/docker-containers-top-7-benefits/ [Accedido: feb-2022].

[92] SentinelOne. *Horizontal Scalability in Your Software: The What, Why, and How*. 2021 [En línea] Disponible en: https://www.sentinelone.com/blog/horizontal-scalability-software/ [Accedido: abr-2022].

[93] theoephraim. *google-spreadsheet*. [En línea] Disponible en: https://theoephraim.github.io/node-google-spreadsheet/#/ [Accedido: abr-2022].

[94] google. *Google Cloud Platform*. [En línea] Disponible en: https://cloud.google.com/?hl=es [Accedido: abr-2022].

[95] . *Números pseudoaleatorios*. [En línea] Disponible en: https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html [Accedido: may-2022].

[96] Loom Network. *Página web para aprender Solidity*. [En línea] Disponible en: https://cryptozombies.io/es/ [Accedido: oct-2021].

[97] Vitalik Buterin. " Coste en gas de cada operación ". En: ().

[98] Juan Blanco. *Extensión de Solidity para VSCode*. [En línea] Disponible en: https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity [Accedido: nov-2021].

[99] Bit2MeAcademy. *Seed Phrase*. [En línea] Disponible en: https://academy.bit2me.com/que-es-frase-semilla-seed-phrase/ [Accedido: nov-2021].

[100] Dotenv Org. *dotenv*. [En línea] Disponible en: https://www.npmjs.com/package/dotenv [Accedido: nov-2021].

[101] MDN Contributors. *Control de acceso HTTP (CORS)*. [En línea] Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/CORS [Accedido: dec-2021].

[102] IBM. *Archivo .env*. [En línea] Disponible en: https://www.ibm.com/docs/es/aix/7.2?topic=files-env-file [Accedido: nov-2021].

[103] MongoDB Inc. *What is MongoDB Atlas?* [En línea] Disponible en: https://www.mongodb.com/docs/atlas/ [Accedido: dic-2021].

[104] B. Damodaran, S. Shirin y S. M. Vargese. "Performance evaluation of MySQL and MongoDB databases". En: *Int. J. Cybern. Inform.(IJCI)* 5 (2016. [En línea] Disponible en: https://cutt.ly/JHtKrJo [Accedido: feb-2022]).

[105] Christopher Ou. *Lite Profile Fields*. 2021 [En línea] Disponible en: https://docs.microsoft.com/en-us/linkedin/shared/references/v2/profile/lite-profile [Accedido: dic-2021].

[106] abarron-linkedin, Deeksha-ramesh y kheiss-linkedin. *URNs and IDs*. 2022 [En línea] Disponible en: https://docs.microsoft.com/en-us/linkedin/shared/api-guide/concepts/urns [Accedido: may-2022].

[107] hylo-li, alexbuckgit y abarron-linkedin. *URNs and IDs*. 2021 [En línea] Disponible en: https://docs.microsoft.com/en-us/linkedin/shared/references/v2/urn-namespace [Accedido: ene-2022].

[108] hylo-li, VSC-Service-Account, kamadolli-linkedin et al. *Profile API*. 2021 [En línea] Disponible en: https://docs.microsoft.com/en-us/linkedin/shared/integrations/people/profile-api [Accedido: ene-2022].

[109] Klughammer. *node-randomstring*. [En línea] Disponible en: https://www.npmjs.com/package/randomstring [Accedido: ene-2022].

[110] T. Fandakly y N. Caporusso. "Beyond passwords: enforcing username security as the first line of defense". En: *International Conference on Applied Human Factors and Ergonomics*. Springer. 2019, 48-58 [En línea] Disponible en: https://link.springer.com/chapter/10.1007/978-3-030-20488-4_5 [Accedido: ene-2022].

[111] Express. *express-session*. [En línea] Disponible en: https://www.npmjs.com/package/express-session [Accedido: ene-2022].

[112] MDN Contributors. *HTTP cookies*. [En línea] Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Cookies [Accedido: dec-2021].

[113] KirstenS. *Cross Site Scripting (XSS)*. [En línea] Disponible en: https://owasp.org/www-community/attacks/xss/ [Accedido: dec-2021].

[114] D. Hardt, Ed. *The OAuth 2.0 Authorization Framework*. RFC 6749. RFC Editor, oct. de 2012. [En línea] Disponible en: https://www.rfc-editor.org/info/rfc6749 [Accedido: ene-2022].

[115] Jared Hanson. *Passport*. [En línea] Disponible en: https://www.npmjs.com/package/passport [Accedido: ene-2022].

[116] Auth0. *Passport LinkedIn OAuth2*. [En línea] Disponible en: https://www.passportjs.org/packages/passport-linkedin-oauth2/ [Accedido: ene-2022].

[117] Sam Bernheim. *PassportJS — The Confusing Parts Explained*. 2018 [En línea] Disponible en: https://hackernoon.com/passportjs-the-confusing-parts-explained-edca874ebead [Accedido: ene-2022].

[118] A. Guerrero, M. V. Buitrago y M. de los Ángeles Curieses. *Estadística básica*. ITM, 2007, 69–70 [En línea] Disponible en: https://cutt.ly/LHyydAz [Accedido: may-2022].

[119] libzy. *material-table*. [En línea] Disponible en: https://material-table.com/#/ [Accedido: abr-2022].

[120] Fco Javier Rubio Arribas. "Factores sociológicos de" la discriminación sociolaboral". En: *Nómadas. Critical Journal of Social and Juridical Sciences* 35.3 (2012. [En línea] Disponible en: https://www.redalyc.org/pdf/181/18126372010.pdf [Accedido: may-2022]).

[121] Brian T McMahon y Linda R Shaw. "Workplace discrimination and disability". En: *Journal of Vocational Rehabilitation* 23.3 (2005. [En línea] Disponible en: https://cutt.ly/LJwfcAR [Accedido: may-2022]), págs. 137-143.

[122] Donna Bobbitt-Zeher. "Gender discrimination at work: Connecting gender stereotypes, institutional policies, and gender composition of workplace". En: *Gender & Society* 25.6 (2011. [En línea] Disponible en: https://cutt.ly/PJwf2Af [Accedido: may-2022]), págs. 764-786.

[123] Emir Ozeren. "Sexual orientation discrimination in the workplace: A systematic review of literature". En: *Procedia-Social and Behavioral Sciences* 109 (2014. [En línea] Disponible en: https://www.sciencedirect.com/science/article/pii/S187704281305252X [Accedido: may-2022]), págs. 1203-1215.

[124] L. M. Shore y C. B. Goldberg. *Age discrimination in the workplace*. Psychology Press, pp 231–254. 2013. [En línea] Disponible en: https://cutt.ly/0JwgbDN [Accedido: may-2022].

[125] C. Huang y B. H. Kleiner. "New developments concerning religious discrimination in the workplace". En: *International Journal of Sociology and Social Policy* (2001. [En línea] Disponible en: https://www.emerald.com/insight/content/doi/10.1108/01443330110789880/full/html [Accedido: may-2022]).

[126] E. C. Hirsh y S. Kornrich. "The context of discrimination: Workplace conditions, institutional environments, and sex and race discrimination charges". En: *American journal of sociology* 113.5 (2008. [En línea] Disponible en: https://www.journals.uchicago.edu/doi/abs/10.1086/525510 [Accedido: may-2022]), págs. 1394-1432.

[127] Infura Inc. *An Introduction to IPFS*. 2020 [En línea] Disponible en: https://blog.infura.io/post/an-introduction-to-ipfs [Accedido: abr-2022].

[128] Infura Inc. *Part 2: Getting Started with Infura's IPFS Service*. 2021 [En línea] Disponible en: https://blog.infura.io/post/getting-started-with-infuras-ipfs-service-updated-june-2021 [Accedido: abr-2022].

[129] IPFS. *Getting started*. [En línea] Disponible en: https://docs.ipfs.io/reference/http/api/#getting-started [Accedido: abr-2022].

[130] daviddias, achingbrain, alanshaw y hugormrdias. *ipfs-http-client*. [En línea] Disponible en: https://www.npmjs.com/package/ipfs-http-client [Accedido: abr-2022].

[131] Ethereum Book. *Tokens*. [En línea] Disponible en: https://github.com/ethereumbook/ethereumbook/blob/develop/10tokens.asciidoc [Accedido: ene-2022].

[132] Kaven Choi. *Comparison for fungible and non-fungible tokens*. 2018 [En línea] Disponible en: https://medium.com/etherscan-blog/tracking-erc-721-non-fungible-token-on-etherscan-b4da1ed955ed [Accedido: may-2022].

[133] Jab Esber. *Reputation vs Tokens*. [En línea] Disponible en: https://future.a16z.com/reputation-based-systems/ [Accedido: mar-2022].

[134] Ethereum. *Ethereum Improvement Proposals*. [En línea] Disponible en: https://eips.ethereum.org/all [Accedido: feb-2022].

[135] OpenZeppelin. *OpenZeppelin Contracts*. [En línea] Disponible en: https://github.com/OpenZeppelin/openzeppelin-contracts [Accedido: feb-2022].

[136] Daniel Tedesco. *EIP-4974: Experience (EXP) Token Standard [DRAFT]*. [En línea] Disponible en: https://ethereum-magicians.org/t/eip-4974-fungible-non-tradable-tokens-or-exp/8805 [Accedido: feb-2022].

[137] Daniel Tedesco. *EIP-4974: Experience (EXP) Token Standard [DRAFT]*. [En línea] Disponible en: https://github.com/ethereum/EIPs/commit/cd3870a788d2386051293df4a0525090182ded51 [Accedido: feb-2022].

[138] StackExchange. *File import callback not supported*. [En línea] Disponible en: https://stackoverflow.com/questions/67321111/file-import-callback-not-supported [Accedido: abr-2022].

[139] Glassdoor. *Glassdoor*. [En línea] Disponible en: https://www.glassdoor.es/member/home/companies.htm [Accedido: mar-2022].

[140] Indeed. *Indeed*. [En línea] Disponible en: https://es.indeed.com/companies?from=gnav-jobsearch--jasx [Accedido: feb-2022].

[141] Wikipedia. *Media ponderada*. [En línea] Disponible en: https://es.wikipedia.org/wiki/Media_ponderada [Accedido: feb-2022].

[142] jdesboeufs y mingchuno. *connect-mongo*. [En línea] Disponible en: https://www.npmjs.com/package/connect-mongo [Accedido: abr-2022].

[143] Leonard Richardson. *Beautiful Soup Documentation*. [En línea] Disponible en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [Accedido: abr-2022].

[144] MongoDB Inc. *mongo_client – Tools for connecting to MongoDB*. [En línea] Disponible en: https://pymongo.readthedocs.io/en/stable/api/pymongo/mongo_client.html#pymongo.mongo_client.MongoClient [Accedido: abr-2022].

[145] Surnet. *swagger-jsdoc*. [En línea] Disponible en: https://www.npmjs.com/package/swagger-jsdoc [Accedido: abr-2022].

[146] Scottie1984. *Swagger UI Express*. [En línea] Disponible en: https://www.npmjs.com/package/swagger-ui-express [Accedido: abr-2022].

[147] SmartBear Software. *Basic Structure*. [En línea] Disponible en: https://swagger.io/docs/specification/basic-structure/ [Accedido: abr-2022].

[148] SmartBear Software. *Data Models (Schemas)*. [En línea] Disponible en: https://swagger.io/docs/specification/data-models/ [Accedido: abr-2022].

[149] Docker. *Dockerfile reference*. [En línea] Disponible en: https://docs.docker.com/engine/reference/builder/ [Accedido: abr-2022].

[150] Natanael Copa. *alpine*. [En línea] Disponible en: https://hub.docker.com/_/alpine [Accedido: abr-2022].

[151] Docker. *Child commands*. [En línea] Disponible en: https://docs.docker.com/engine/reference/commandline/docker/ [Accedido: abr-2022].

[152] Docker. *Compose specification*. [En línea] Disponible en: https://docs.docker.com/compose/compose-file/ [Accedido: abr-2022].

[153] Nginx. *What Is Load Balancing?* [En línea] Disponible en: https://www.nginx.com/resources/glossary/load-balancing/ [Accedido: may-2022].

[154] Docker. *Docker Hub*. [En línea] Disponible en: https://www.docker.com/products/docker-hub/ [Accedido: abr-2022].

[155] Nginx. *Module ngx_http_upstream_module*. [En línea] Disponible en: https://nginx.org/en/docs/http/ngx_http_upstream_module.html [Accedido: may-2022].

[156] Semantic UI. *Semantic UI*. [En línea] Disponible en: https://semantic-ui.com/ [Accedido: abr-2022].

[157] Recharts Group. *Recharts*. 2018 [En línea] Disponible en: https://recharts.org/en-US/ [Accedido: abr-2022].

[158] Ryan McNierney. *Using React + Google Sheets as your CMS*. 2018 [En línea] Disponible en: https://medium.com/@ryan.mcnierney/using-react-google-sheets-as-your-cms-294c02561d59 [Accedido: abr-2022].

[159] gilbara. *React Joyride*. [En línea] Disponible en: https://github.com/gilbarbara/react-joyride [Accedido: abr-2022].

[160] Refsnes Data. *Window localStorage*. [En línea] Disponible en: https://www.w3schools.com/jsref/prop_win_localstorage.asp [Accedido: abr-2022].

[161] Mosh Hamedani. *How to use localStorage with React*. 2019 [En línea] Disponible en: https://programmingwithmosh.com/react/localstorage-react/ [Accedido: abr-2022].

[162] *OpenAI API Reference*. 2020 [En línea] Disponible en: https://beta.openai.com/docs/api-reference/ [Accedido: mayo-2021].

[163] Etherscan. *Etherscan*. [En línea] Disponible en: https://etherscan.io/ [Accedido: may-2022].

[164] VanwaTech. *View file stored in IPFS*. [En línea] Disponible en: https://ipfsbrowser.com/ [Accedido: may-2022].

[165] Digiconomist. *Ethereum Energy Consumption Index*. [En línea] Disponible en: https://digiconomist.net/ethereum-energy-consumption [Accedido: may-2022].

[166] Ethereum. *Crear una EIP*. [En línea] Disponible en: https://github.com/ethereum/EIPs [Accedido: may-2022].

[167] RAILGUN Project. *What are zk-SNARKs? A Simple Guide*. 2021 [En línea] Disponible en: https://medium.com/@Railgun_Project/what-are-zk-snarks-a-simple-guide-221734766cee [Accedido: may-2022].

[168] Oracle. *Conceptos de Container Engine y Kubernetes*. [En línea] Disponible en: https://docs.oracle.com/es-ww/iaas/Content/ContEng/Concepts/contengclustersnodes.htm [Accedido: may-2022].

[169] Overview. *The MITRE Corporation*. [En línea] Disponible en: https://www.cve.org/About/Overview [Accedido: may-2022].

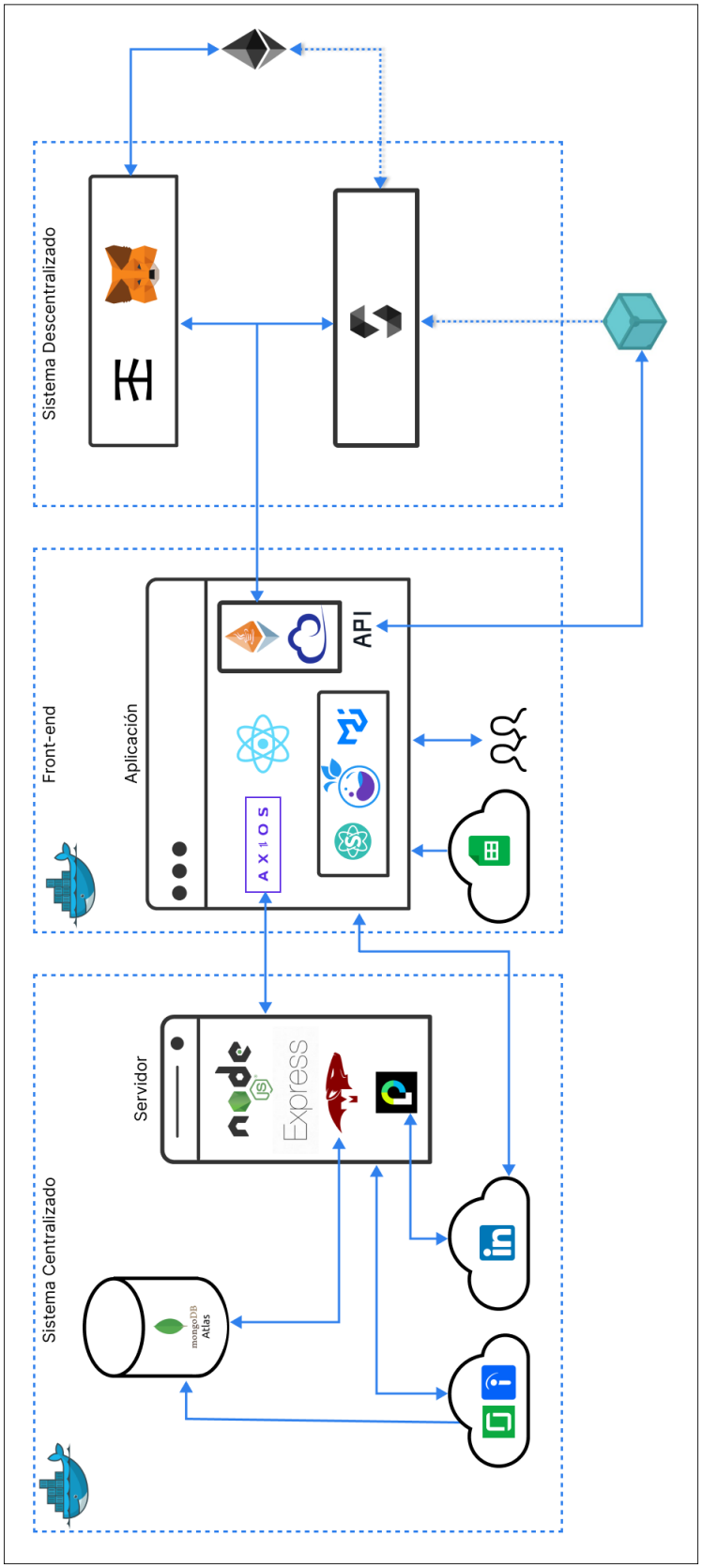
[170] OWASP. *Who is the OWASP Foundation?* [En línea] Disponible en: https://owasp.org/ [Accedido: may-2022].

[171] INCIBE. *Qué es INCIBE-CERT*. [En línea] Disponible en: https://www.incibe-cert.es/sobre-incibe-cert/que-es-incibe-cert [Accedido: may-2022].

[172] W3C. *How to Meet WCAG (Quick Reference)*. 2019 [En línea] Disponible en: https://www.w3.org/WAI/WCAG21/quickref/#adaptable [Accedido: may-2022].

Apéndice

A. Arquitectura



B. Smart-Contracts

B.1. Versión Inicial

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <= 0.8.10;
pragma experimental ABIEncoderV2;

contract ReputationControl {

    //Informacion de denuncias
    struct complaint {
        string user;
        string text;
        string dateC;
        string typeC;
        string gender;
        string relation;
        bool consent;
    }

    //Informacion de la empresa -> denuncias + reputacion
    //La reputacion seria 100 - ndenuncias recibidas
    struct companyData{
        uint reputation;
        bool participating;
        complaint[] complaints;
    }

    //Nombres de las empresas que se pueden denunciar para poder iterar en el
        mapping
    string[] private companies = new string[](0);

    //Mapping que devuelve los datos de las denuncias referentes a una empresa
    mapping(string => companyData) private companyMapping;

    //Propietario del sistema
    address immutable owner;

    modifier isNotCompany(string memory c_name){
        require(!companyMapping[c_name].participating, "This company doesn't exist
            in the system");
        _;
    }

    modifier isCompany(string memory c_name){
        require(companyMapping[c_name].participating, "This company doesn't exist
            in the system");
        _;
    }

    modifier onlyOwner{
        require(msg.sender == owner, "No allowances");
        _;
    }

    //Se inicializa la reputacion
    constructor(string[] memory _companies) {
        companies = _companies;
        owner = msg.sender;
```

```
54          for(uint i = 0; i < companies.length ; i++){ //Damos valor inicia
55              companyMapping[companies[i]].reputation = 100;
56              companyMapping[companies[i]].participating = true;
57          }
58      }
59
60      //Function para devolver las empresas
61      function getCompanies() external view returns (string[] memory){
62          return companies;
63      }
64
65      //Funcion para devolver los datos de la empresa: Denuncias, futuras metricas
66      function getComplaints(string memory c_name) external view returns (complaint
            [] memory) {
67          return companyMapping[c_name].complaints;
68      }
69
70      //Funcion para anadir empresas
71      function newCompany(string memory c_name) external isNotCompany(c_name)
            onlyOwner{
72          companies.push(c_name);
73          companyMapping[c_name].reputation = 100;
74          companyMapping[c_name].participating = true;
75      }
76
77      //Funcion para obtener todas las reputaciones
78      function getAllReputations() external view returns (uint[] memory) {
79          uint[] memory reputations = new uint[](companies.length);
80          for(uint i=0; i< companies.length; i++){
81              reputations[i] = (companyMapping[companies[i]].reputation);
82          }
83          return reputations;
84      }
85
86      function getReputations(string[] memory c_names) external view returns (uint[]
             memory){
87          uint[] memory reputations = new uint[](c_names.length);
88          for(uint i=0; i< c_names.length; i++){
89              if(companyMapping[c_names[i]].participating)
90                  reputations[i] = (companyMapping[c_names[i]].reputation);
91              else reputations[i] = 101; //Valor no valido, captar en front-end
92          }
93          return reputations;
94      }
95
96      //Funcion para insertar denuncia
97      function newComplaint(string memory c_name, string memory u_name, string
            memory _text, string memory _date, string memory _type, string memory
            _gender, string memory _relation, bool _consent) external isCompany(c_name)
             onlyOwner{
98          companyMapping[c_name].complaints.push(complaint(u_name, _text, _date,
                _type, _gender, _relation, _consent));
99          uint ret = companyMapping[c_name].reputation;
100         if(ret > 0) companyMapping[c_name].reputation -= 1; //0-1 en uint da un
                valor positivo enorme, no -1.
101     }
102 }
```

Volver a lectura

B.2. Versión Definitiva

B.2.1. Versión ERC20

```
// SPDX-License-Identifier: CC0
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ReksTokens is ERC20{

    address immutable owner;
    modifier onlyOwner{
        require(msg.sender == owner, "No permission");
        _;
    }

    constructor(string memory _name, string memory _symbol) ERC20(_name, _symbol){
        owner = msg.sender;
    }

    function mint(address to, uint256 amount) public onlyOwner{
        _mint(to,amount);
    }

    function burn(address to, uint256 amount) public onlyOwner{
        _burn(to,amount);
    }

    function transfer(address to, uint256 amount) public override onlyOwner
        returns (bool) {
        address _owner = _msgSender();
        _transfer(_owner, to, amount);
        return true;
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public override onlyOwner returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }

}

contract ReputationControl {

    struct CompanyData{
        address _address;
        //Posicion de la denuncia en el array (+1). Se utiliza para eliminar la
            denuncia
        uint pos;
        mapping(string => uint) posArray;
        string[] hashComplaints;
    }

```

```
55      ReksTokens tokenContract;
56      mapping(string => CompanyData) infoCompanies;
57      string[] companies;
58      address immutable owner;
59      string amountDonated;
60
61       modifier onlyOwner{
62          require(msg.sender == owner,"No allowances");
63          _;
64      }
65
66      modifier isCompany(string memory _company){
67          require(infoCompanies[_company]._address != address(0), "Not detected as a
                participant in our system");
68          _;
69      }
70
71      constructor(){
72          owner = msg.sender;
73          tokenContract = new ReksTokens("Reks", "R");
74          amountDonated = "0";
75      }
76
77
78      function newCompanies(string[] memory _companies) external onlyOwner{
79          for(uint i = 0; i < _companies.length; ++i){
80              if(infoCompanies[_companies[i]]._address == address(0)){
81                  address add = address(bytes20(keccak256(abi.encodePacked(
                        _companies[i]))));
82                  infoCompanies[_companies[i]]._address = add;
83                  infoCompanies[_companies[i]].pos = companies.length;
84                  companies.push(_companies[i]);
85              }
86          }
87      }
88
89      function newCompany(string memory _company) external onlyOwner{
90          require(infoCompanies[_company]._address == address(0),"Company is already
                in the system");
91          address add = address(bytes20(keccak256(abi.encodePacked(_company))));
92          infoCompanies[_company]._address = add;
93          infoCompanies[_company].pos = companies.length;
94          companies.push(_company);
95      }
96
97      function retireCompany(string memory _company) external onlyOwner isCompany(
            _company){
98          address account = infoCompanies[_company]._address;
99          tokenContract.burn(account, tokenContract.balanceOf(account));
100         uint index = infoCompanies[_company].pos;
101         string memory lastCompany = companies[companies.length - 1];
102         companies[index] = lastCompany;
103         infoCompanies[lastCompany].pos = index;
104         companies.pop();
105         delete infoCompanies[_company];
106     }
107
108     function getTotalComplaints() external view returns(uint){
109         return tokenContract.totalSupply();
110     }
```

```
111
112     function getCompaniesNames() external view returns (string[] memory){
113         return companies;
114     }
115
116     function getReputation(string memory _company) external view isCompany(
            _company) returns (uint){
117         return tokenContract.balanceOf(infoCompanies[_company]._address);
118     }
119
120     function getAllReputations() external view returns (uint[] memory){
121         uint [] memory _balances = new uint[](companies.length);
122         for(uint i = 0; i<companies.length;++i){
123             _balances[i] = tokenContract.balanceOf(infoCompanies[companies[i]].
                    _address);
124         }
125         return _balances;
126     }
127
128     function getCompanyAddress(string memory _company) external view isCompany(
            _company) returns(address){
129         return infoCompanies[_company]._address;
130     }
131
132     function getCompaniesAddresses() external view returns (address[] memory){
133         uint l = companies.length;
134         address [] memory accounts  = new address[](l);
135         for(uint i = 0; i < l; ++i){
136             accounts[i] = infoCompanies[companies[i]]._address;
137         }
138         return accounts;
139     }
140
141     function getCompanyComplaints(string memory _company) external view isCompany(
            _company) returns (string[] memory){
142         return infoCompanies[_company].hashComplaints;
143     }
144
145     function newComplaint(string memory _company, string memory _hash) external
            isCompany(_company) onlyOwner{
146         infoCompanies[_company].hashComplaints.push(_hash);
147         infoCompanies[_company].posArray[_hash] = infoCompanies[_company].
                hashComplaints.length;
148         tokenContract.mint(infoCompanies[_company]._address,1);
149     }
150
151     function retireComplaint(string memory _company, string memory _hash) external
             isCompany(_company) onlyOwner{
152         uint index = infoCompanies[_company].posArray[_hash];
153         require(index>0, "This complaint isn't registered");
154         //Intercambiar esta denuncia con ultimo elemento del array
155         uint last = infoCompanies[_company].hashComplaints.length - 1;
156         string memory lastHash = infoCompanies[_company].hashComplaints[last];
157         infoCompanies[_company].posArray[lastHash] = index;
158         infoCompanies[_company].posArray[_hash] = 0;
159         infoCompanies[_company].hashComplaints[index - 1] = lastHash;
160         infoCompanies[_company].hashComplaints.pop();
161         //Quemar el token de esa denuncia
162         tokenContract.burn(infoCompanies[_company]._address,1);
163     }
```

```
164
165     function changeAmountDonated(string memory amount) external onlyOwner{
166         amountDonated = amount;
167     }
168
169     function getAmountDonated() external view returns (string memory){
170         return amountDonated;
171     }
172 }
```

Volver a lectura

B.2.2. Versión ERC4974

```
1  // SPDX-License-Identifier: CC0
2  pragma solidity ^0.8.0;
3
4  import "./Auxiliar/ERC4974.sol";
5
6  contract ReputationControl {
7
8      struct CompanyData{
9          address _address;
10         //Posicion de la denuncia en el array (+1). Se utiliza para eliminar la
               denuncia
11         uint pos;
12         mapping(string => uint) posArray;
13         string[] hashComplaints;
14     }
15
16     ERC4974 tokenContract;
17     mapping(string => CompanyData) infoCompanies;
18     string[] companies;
19     address immutable owner;
20     string amountDonated;
21
22     modifier onlyOwner{
23         require(msg.sender == owner,"No allowances");
24         _;
25     }
26
27     modifier isCompany(string memory c_name){
28         require(infoCompanies[c_name]._address != address(0), "Not detected as a
               participant in our system");
29         _;
30     }
31
32     constructor(){
33         owner = msg.sender;
34         tokenContract = new ERC4974("Reks", "R", address(this));
35         amountDonated = "0";
36     }
37
38
39     //Aniadir empresas
40     function newCompanies(string[] memory _companies) external onlyOwner{
41         for(uint i = 0; i < _companies.length; ++i){
42             if(infoCompanies[_companies[i]]._address == address(0)){
43                 address add = address(bytes20(keccak256(abi.encodePacked(
                       _companies[i]))));
```

```
                infoCompanies[_companies[i]]._address = add;
                infoCompanies[_companies[i]].pos = companies.length;
                companies.push(_companies[i]);
                tokenContract.setParticipation(add,true);
            }
        }
    }

    function newCompany(string memory _company) external onlyOwner{
        require(infoCompanies[_company]._address == address(0),"Already in the
            system");
        address add = address(bytes20(keccak256(abi.encodePacked(_company))));
        infoCompanies[_company]._address = add;
        infoCompanies[_company].pos = companies.length;
        companies.push(_company);
        tokenContract.setParticipation(add,true);
    }

    function retireCompany(string memory _company) external onlyOwner isCompany(
        _company){
        address account = infoCompanies[_company]._address;
        tokenContract.transfer(account, address(0), tokenContract.balanceOf(
            account));
        uint index = infoCompanies[_company].pos;
        string memory lastCompany = companies[companies.length - 1];
        companies[index] = lastCompany;
        infoCompanies[lastCompany].pos = index;
        companies.pop();
        delete infoCompanies[_company];
    }

    function getTotalComplaints() external view returns(uint){
        return tokenContract.totalSupply();
    }

    function getCompaniesNames() external view returns (string[] memory){
        return companies;
    }

    function getReputation(string memory _company) external view isCompany(
        _company) returns (uint){
        return tokenContract.balanceOf(infoCompanies[_company]._address);
    }

    function getAllReputations() external view returns (uint[] memory){
        uint [] memory _balances = new uint[](companies.length);
        for(uint i = 0; i<companies.length;++i){
            _balances[i] = tokenContract.balanceOf(infoCompanies[companies[i]].
                _address);
        }
        return _balances;
    }

    function getCompanyAddress(string memory _company) external view isCompany(
        _company) returns(address){
        return infoCompanies[_company]._address;
    }

    function getCompaniesAddresses() external view returns (address[] memory){
        uint l = companies.length;
```

```
98          address [] memory accounts  = new address[](1);
99          for(uint i = 0; i < 1; ++i){
100             accounts[i] = infoCompanies[companies[i]]._address;
101         }
102         return accounts;
103     }
104
105     function getCompanyComplaints(string memory _company) external view isCompany(
            _company) returns (string[] memory){
106         return infoCompanies[_company].hashComplaints;
107     }
108
109     function newComplaint(string memory _company, string memory _hash) external
            isCompany(_company) onlyOwner{
110         infoCompanies[_company].hashComplaints.push(_hash);
111         infoCompanies[_company].posArray[_hash] = infoCompanies[_company].
                hashComplaints.length;
112         tokenContract.transfer(address(0),infoCompanies[_company]._address,1);
113     }
114
115     function retireComplaint(string memory _company, string memory _hash) external
             isCompany(_company) onlyOwner{
116         uint index = infoCompanies[_company].posArray[_hash];
117         require(index>0, "This complaint isn't registered");
118         //Intercambiar esta denuncia con ultimo elemento del array
119         uint last = infoCompanies[_company].hashComplaints.length - 1;
120         string memory lastHash = infoCompanies[_company].hashComplaints[last];
121         infoCompanies[_company].posArray[lastHash] = index;
122         infoCompanies[_company].posArray[_hash] = 0;
123         infoCompanies[_company].hashComplaints[index - 1] = lastHash;
124         infoCompanies[_company].hashComplaints.pop();
125         //Quemar el token de esa denuncia
126         tokenContract.transfer(infoCompanies[_company]._address,address(0),1);
127     }
128
129     function changeAmountDonated(string memory amount) external onlyOwner{
130         amountDonated = amount;
131     }
132
133     function getAmountDonated() external view returns (string memory){
134         return amountDonated;
135     }
136 }
```

Volver a lectura

C. Evaluación con usuarios

C.1. Screening

El screening en cuestión se puede encontrar en este enlace [3].

Este breve cuestionario es previo a la evaluación para conocer el nivel tecnológico, afinidad con respecto a la tecnología Blockchain y características generales del ambiente laboral de tu trabajo.

Preguntas dirigidas a la persona En primer lugar preguntamos por la edad dividiendo en cinco rangos que cubren las posibles edades de una persona que a día de hoy esté trabajando. Luego se pregunta por el género.

Preguntas dirigidas a la situación laboral de la persona Para comenzar se clasifica al entrevistado según su situación laboral (trabaja, paro, buscando trabajo). También se le pide que valore la importancia que atribuye al ambiente laboral dentro de la empresa, y si lo tienen en consideración para decidir aceptar un trabajo. Además, se le consulta cual es su experiencia con situaciones de discriminación dentro de la empresa, tanto recibir como presenciar. Para acabar, el usuario realiza unas consideraciones sobre los medios dispuestos para exhibir estas situaciones, si cree que que estos tienen consecuencias positivas y si es viable cambiar el ambiente laboral por medio de estos.

Preguntas dirigidas a la tecnología Se le pregunta al usuario sobre su familiaridad y conocimiento previo sobre la tecnología Blockchain.

C.2. Guión del orientador

Buenos días. Usted ha sido escogido para realizar una prueba de evaluación de nuestro proyecto del TFG. Este es una aplicación centrada en el ámbito laboral, que permite a los usuarios denunciar situaciones de discriminación que hayan sufrido en una empresa. Además, es posible observar las reputaciones de cada compañía, así como todas las denuncias que han recibido.

Antes de comenzar la evaluación, deberás rellenar un pequeño formulario para conocer su nivel tecnológico, su afinidad con la tecnología Blockchain y su ambiente laboral.

Esta prueba consistirá en la realización de 6 tareas en la aplicación, simulando el comportamiento real de cualquier usuario. La página trata de ser lo más intuitiva posible, con un breve tutorial al comienzo para familiarizarse con ella. Se irán enunciando una a una las tareas, cuando la anterior haya sido superada. Es recomendable que vaya diciendo en alto todo lo que se le ocurra, su proceso mental en la navegación y sus opiniones. De igual forma, al final de la prueba agradeceremos todos los comentarios que haga, tanto si son positivos como negativos.

C.3. Descripción de las evaluaciones

Se efectuaron 5 evaluaciones a participantes de diferente índole, siguiendo el guión de la búsqueda de participantes mencionados en el documento. Se pretende entrevistar a personas con diferentes

[3] https://docs.google.com/forms/d/e/1FAIpQLSeyrAmFmKGZGSvQ-Ob5egNuSFHAJPf2nC6qOmqGi8qpbmVdkA/viewform

experiencias y puntos de vista para obtener una amplia variedad de respuestas y acciones en las evaluaciones. En esta sección se describe lo que sucedió en cada una de ellas.

Participante 1
Persona de 23 años que se encuentra realizando prácticas en empresa, por lo que le interesa observar su reputación en caso de desear quedarse. Es una persona que tiene interés por la temática social ya que ella ha sufrido comentarios discriminatorios previamente. Ha escuchado previamente sobre la tecnología Blockchain pero no posee conocimientos sobre el tema.

Una vez explicado el propósito de la entrevista, completó el formulario correspondiente al screening.Es una persona muy detallista y curiosa, comenzando a examinar la aplicación nada más acceder a esta.

En primer lugar se le indicó que siguiera el onboarding, el cuál pudo seguir sin problema. Comentó que no comprendía los círculos entre la palabra de Denuncia aunque valoró positivamente la presencia de la este. Detalló que el home es fácil de comprender pero le gustaría que hubiera menos módulos para no distraerse y que las categorías de las gráficas estuvieran más separadas. Adicionalmente preguntó sobre la cantidad recaudada ya que no conocía la abreviatura ETH, la cuál fue explicada por el entrevistador. No obstante, no presentó ningún problema en explorar cada rincón, solo realizando comentarios sobre la estética.

Al acceder a una empresa concreta, mencionó que la página de empresa era muy intuitiva y que se comprendía correctamente. Navegó por las gráficas sin problemas y leyó las denuncias que tenían el consentimiento activado.

Posteriormente se indicó que iniciara sesión mediante LinkedIn. Esta tarea la realizó con gran rapidez y aportó que se comprendía correctamente su función. Tampoco presentó problemas con el formulario de denuncia, aunque el entrevistador tuvo que clarificar que no todos los campos eran obligatorios. El diseño le pareció claro y directo, permitiendo al usuario centrarse en una sola acción.
Visitó la página de información del proyecto sin ser guiada y probó a realizar una donación, indicando el entrevistador que se realizaría mediante la cuenta puesta a su disposición.

Finalmente, mencionó que debería haber menos cambios de color. Concluyó que la aplicación era intuitiva y fácilmente navegable; estando las funcionalidades posicionadas de forma accesible y cohesionada.

Participante 2
Persona de entre 25 y 30 años que, actualmente, están trabajando en una empresa privada y estaría interesada en denunciar si pasase algo discriminatorio.

El participante tiene las cosas muy claras y es muy seguro de si mismo. Trabaja en una empresa privada desde hace 5 años y sabe que si se diese una situación de discriminación emplearía los medios necesarios para denunciarla. Al hablarle de nuestra plataforma, le parece muy interesante. Cree mucho en el talento joven y en propulsar este tipo de iniciativas mediante la confianza de los usuarios. Tiene información sobre Blockchain porque trabaja en el departamento tecnológico de su empresa y es curioso y le gusta informarse sobre temas de actualidad. Nunca ha empleado una plataforma para denunciar.

Una vez terminó el screening, se pasó a mostrar la plataforma. El tour virtual le sorprendió mu-

cho, dijo que le parecía algo que aportaba mucho valor y que los usuarios nuevos podían aprender a usar de forma fácil, tanto si saben de tecnología como si no. Sobre la pantalla principal, dijo que le parecía muy estética y que le parecía muy origina usar un carousel para mostrar las empresas. Dijo que le faltaba algún tipo de elemento donde pudiese buscar una empresa determinada sin tener que recorrer todo el carrousel. Le gustó mucho la idea de mostrar los tops, pero preguntó que en qué nos basábamos para calcular la reputación, que no sabía cómo verlo. Miró las gráficas y dijo que le parecían muy visuales y que aportaban fiabilidad. A continuación, dijo que la funcionalidad principal de la plataforma, denunciar, estaba a un solo clic.

Hizo clic en la empresa Telefónica, para ello, tuvo que esperar a que pasase de nuevo el carousel, sin hacer uso de los tres puntos de abajo.

Una vez estaba en la página de la empresa, dijo que el diseño casaba completamente con la otra pantalla. Además, le gustó que también se pudiese denunciar desde esta página, sin tener que volver a la página principal. Interactuó con la tabla, tras un tiempo, entendió que se podía leer la denuncia, y pulsó el botón en los casos en que estaba en verde, e interactuó con el dialog correctamente. Intentó pulsar en aquellas en las que no se leía la experiencia (estaba en rojo) para ver que sucedía, y vio que nada. A la hora de usar las gráficas, le pareció una buena idea usar pestañas para no tener todo el espacio lleno de gráficas. Intentó pulsar en varias de ellas , cuando llegó a una de las que quedan deshabilitadas por no tener datos suficientes, no entendía por qué no funcionaba.

Para registrarse en LinkedIn, como estaba acostumbrada a usar muchas páginas en las que tiene que registrarse con normalidad, le pareció sencillo.

Volvió a la página principal para denunciar y le sorprendió que no volviese a salir el tour virtual. Le comentó el entrevistador que solo aparece una vez, y le pareció una buena idea, sino resultaría tedioso. Hizo click en el botón de denunciar que aparece en la página principal. Rellenó el formulario de denuncia en orden, por mobbing. Comentó que las preguntas con opción múltiple desplegable le parecían una buena idea. Respecto a los datos, dijo que sí, pero que echaba en falta en algún lugar una pantalla con información sobre todo el proyecto. El examinador le dijo que eso quedaba explicado en una pantalla a la que se accedía desde el índice. Dijo que esto le parecía poco útil, porque si alguien no encontraba el índice, no conocería nada de la información sobre la plataforma.

Al terminar las tareas, dijo que le apetecía ver la página con la información de la empresa. Entró en ella, tardando un tiempo en encontrar el índice, y le encantó la idea de donar y de emplear un texto motivador para ello. Sin embargo, nos preguntó que cómo podría ayudar a donar alguien que no tuviese idea sobre la Blockchain. Interactuó con la página, pulsando en las empresas y viendo que desde ellas se podía acceder a la página real de la empresa. Le gustó mucho.

Al acabar, felicitó al entrevistador y le dijo que en términos de diseño y uso estaba muy bien, y no se necesitaban tener conocimientos previos de Blockchain para emplearla, punto a favor, bajo su punto de vista.

Participante 3

El participante a evaluar tiene 31 años y a día de hoy se encuentra trabajando en el sector privado. Hace unos meses se le obligó a trasladarse a otra sede de la empresa en el extranjero. Relata su descontento con el ambiente actual y muestra su interés por regresar a su anterior sede o incluso

abandonar su puesto. En los pocos meses que lleva ha observado algún caso de discriminación y aun haciéndoselo saber a sus superiores no ha visto reflejada ninguna medida que evitara esos problemas. Sin embargo, confirma creer que si se establecen las acciones necesarias sí es posible cambiar los comportamientos negativos dentro de la empresa. Prefirió no señalar el género que lo identifica, pues asegura no suele indicarlo en este tipo de cuestionarios. Conoce muy poco sobre los fundamentos de la tecnología Blockchain, aunque esto no se traslada a su interés por las criptomonedas, pues siempre ha puesto interés en la inversión y comenta que sigue muy de cerca algunos proyectos como Solana o Cardano.

Después se inició la prueba, comenzando con el onboarding le resultó muy útil y declaró echar en falta este tipo de esquemas en otras aplicaciones y páginas web. Sin embargo, le costaba un poco leerlo pues la letra era algo pequeña.

Le resultó sencillo entender la información disponible, en parte por el tutorial previo. A su vez, muestra interés por los datos recogidos, pues afirma desconocer otra plataforma que los exponga de esta forma.

Se encontró con problemas para acceder a una empresa en concreto, en vez de acceder desde el carousel o los tops, se dirigió a la pagina Compañía y después a la de información, que te redirige a la página oficial de la empresa y no a la informativa. El evaluador tuvo que intervenir para indicarle como acceder. Señalaba que era necesaria una página a modo de buscador de empresas. A continuación, vió la información sobre denuncias y las gráficas e igual que en la página principal le fue fácil comprenderlo. Valoró positivamente los colores, pues hacían todo más sencillo.

Conectarse con su usuario de LinkedIn para poner una denuncia. (Si no tiene, la evaluación será exclusivamente para observar si la aplicación es intuitiva y los datos que se muestran en ella son entendibles).
No tuvo problemas para iniciar sesión con LikedIn. Pero le pareció un poco pequeño el icono.

Para denunciar a HP accedió a la página de la empresa por lo que le indicó el evaluador al principio, para luego dar a denunciar, cuando puede pulsar en denunciar en la página principal. Señaló que si denunciaba desde una empresa esta debería aparecer ya seleccionada en el campo del formulario. Aprovechó la situación para indicar el caso cercano que le toca, aunque como denuncia por una discriminación presenciada y no recibida, dudaba si dar algún dato del afectado. Acabó rellenando solo los campos obligatorios y marcó la casilla de publicar, puesto que le resulta el atractivo principal de la aplicación y considera que la opción de no mostrar es de alguna manera perjudicial. Cree que debería ser un entorno para expresarse libremente.

Muestra confianza en el proyecto, en gran medida por conocer la gente que hay detrás de este. Como había utilizado antes Metamask le resultó fácil donar, pero recordó que cuando le tocó configurarlo por primera vez tuvo algún que otro problema.

Participante 4

Este participante es una señora de 52 años, la cuál pasó por una situación de discriminación en el pasado, en relación a su género. Al principio expresa sus ganas por participar en la prueba, pues cree que puede ser una buena herramienta para ayudar a personas que estuvieron en su misma situación. Además, expresa que de los Bitcoins solo ha escuchado por la televisión y en conversaciones con amigos. Cree que le queda ya muy lejos, prefiere utilizar dinero real.

Una vez rellenado el Screening, la evaluación comenzó con el tutorial. Le encantó. Dijo que era muy intuitivo y ayudaba bastante al ser la primera vez que utilizaba la app. Acto seguido estuvo un par de minutos observando la página principal, detallando aquello que le parecía extraño y lo que le gustaba. No entendió bien de qué era cada gráfica, echaba de menos leyendas más específicas y además quería ver todas las empresas de la aplicación y no supo como hacerlo. Los aspectos positivos fueron la sencillez de la página, las reputaciones, los tops, poder denunciar y el carrousel queda claro, conciso y estético.

Para ver los datos de Telefónica, espero a que en el carrousel apareciese. Pulsó al botón de ver empresa, aunque expreso que no le convencía la idea porque tenía que pulsar muy rápido. Dentro de la empresa le gustó el esquema, que además se parecía al de la página principal. La tabla de denuncias también le gustó. Tuvo algunas dudas a la hora de leer la denuncia, pues no sabía que había que pulsar el botón. Tras unos segundos lo descubrió. Respecto a las gráficas, en este caso sí entendió a que se referían.

Para donar era necesario que se conectase a LinkedIn. Esta tarea le resultó muy complicada, a pesar de su simpleza. No recordaba el logo de LinkedIn, pues aunque tuviese cuenta nunca lo utiliza. El moderador tuvo que intervenir en esta tarea.

A la hora de denunciar, accedió desde la página de Telefónica. Al leer el diálogo, entendió que sus datos iban a ser públicos, específicamente que cualquiera podía verlos. Rellenó el formulario, entendiendo que tenía que escribir en cada apartado. Al enviar la denuncia, no entendió porqué había que esperar a que estuviera cargada.

Una vez emitida la denuncia, volvió a la página principal y esperaba que la reputación hubiese cambiado. Se decepcionó un poco, pero el moderador le comentó que al principio de la aplicación, la reputación está mayormente basada en otras fuentes externas. Acto seguido fue a la página de la información del proyecto a través del menú desplegable. Esta página también le pareció intuitiva, y encima pudo ver todas las empresas juntas. Comentó que era mejor opción incluir esta funcionalidad en la página principal. Como no conocía nada de Blockchain, la tarea de donar fue simplemente una pregunta de si intuía como donar. No le gustó que ella no pudiese donar.

Al finalizar la prueba, comentó que en líneas generales se había sentido cómoda utilizando la aplicación, le gustaba el diseño y lo veía intuitivo. Además, especificó que no había notado que la web funcionase con Blockchain, salvo los tiempos de espera de emitir la denuncia. Para terminar, rellenó el cuestionario final sin introducir observaciones, pues el moderador ya lo había apuntado todo.

Participante 5

El participante es un hombre de 49 años que trabaja en una empresa mediana. Le pareció interesante la propuesta de nuestra plataforma y nos pareció un buen candidato para evaluarla. Es una persona que se preocupa por los derechos sociales y, aunque nunca ha sufrido ningún caso discriminatorio, considera que hoy en día es difícil controlar ese tipo de situaciones en una empresa y nuestra plataforma puede ser interesante para visibilizar estos hechos desagradables. Además, en caso de estar su empresa en nuestro proyecto, si le interesaría mirar la reputación que esta tiene y el de otras empresas a la hora de considerar si cambiarse de empresa

Rellenó el Screening inicial y consideramos que era una persona adecuada para valorar nuestra aplicación, a pesar de no tener mucho conocimiento sobre la tecnología Blockchain. A pesar de no tener LinkedIn pudo realizar la prueba sin problemas para medir si las acciones y plataforma eran intuitivas.

El onboarding y pantallas principales le gustaron bastante y le facilitó mucho la primera toma de contacto con la aplicación. Quedó agradado pues como a él le gusta mucho lo de ser capaz de ver las reputaciones de las empresas quedó rápidamente complacido. Entendió toda la información que aparecía en el inicio sin mayores problemas y le gustó mucho la estética.

Donde tuvo mayores problemas al inicio fue a la hora de encontrar el botón de donar pues tuvo que probar buscando en el incio antes de darse cuento que se encontraba en la ventana Ïnformación del Proyecto".

Observó sin problemas toda la información de Telefónica, aunque tuvo que esperar a que apareciese la empresa en el carrousel de la pantalla principal pues no intuyó que pinchando desde el Top 5 se podía acceder a la pantalla de la empresa también. Por lo demás, quedó bastante satisfecho con los datos y que se mostraba, así como la información sobre las denuncias realizadas.

Como hemos indicado anteriormente, no tenía LinkedIn no inició sesión, pero aún así se le asignó la tarea de denunciar a una empresa para que rellenase y comentase su valoración sobre el cuestionario. El cuestionario, como ventana, le gustó, pero estuvo reacio a dejar algunos de sus datos porque creía que luego podía dar pistas a la empresa sobre qué persona había denunciado y esta empresa podía actuar en consecuencia.

La acción de donar la realizó sin problemas, pero volvió a comentar que era la acción más difícil de averiguar de primeras al no ser del todo intuitivo cómo encontrar el botón.

Finalmente, comentó que le había gustado mucho la aplicación estéticamente y la finalidad del proyecto. Sin embargo, al no conocer en profundidad la tecnología y ser un tema controvertido, comentó que antes de usarla para denunciar le gustaría investigar un poco para saber si poder fiarse del todo. De todas formas, concluyó que le había parecido muy interesante y que le había gustado mucho y estaba seguro de que la usaría para mirar reputaciones de empresas, incluida la suya, a la hora de buscar un cambio de empresa.