UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Aplicación web para operar con contratos digitales a través de blockchain

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gozalvo Cervera, Francisco Javier

Tutor/a: Esparza Peidro, Javier

CURSO ACADÉMICO: 2022/2023

Resumen

El proyecto aborda la creación de una plataforma web que permita a sus usuarios crear, enviar, modificar y gestionar sus contratos de una forma simple y segura. Además la aplicación permitirá tecnologías propias de la Web 3.0, tales como conexiones a la blockchain y transacciones a la misma, otorgando un punto extra de descentralización, inmutabilidad y transparencia en cuanto a los contratos entre los usuarios.

Palabras clave

Contratos, Aplicación Web, Blockchain, Web 3.0, Smart Contracts

Resum

El projecte aborda la creació d'una plataforma web que permeta als seus usuaris crear, enviar, modificar i gestionar els seus contractes d'una forma simple i segura. A més l'aplicació permetrà tecnologies pròpies de la Web 3.0, com ara connexions a la *blockchain i transaccions a aquesta, atorgant un punt extra de descentralització, inmutabilitat i transparència quant als contractes entre els usuaris.

Paraules clau

Contractes, Aplicació Web, Blockchain, Web 3.0, Smart Contracts

Abstract

The project addresses the creation of a web platform that allows its users to create, send, modify and manage their contracts in a simple and secure way. In addition, the application will allow technologies typical of Web 3.0, such as connections to the blockchain and transactions to it, granting an extra point of decentralization, immutability and transparency in terms of contracts between users.

Keywords

Contracts, Web Application, Blockchain, Web 3.0, Smart Contracts

Agradecimientos

A mi familia, pareja y amigos, quienes me han brindado todo su apoyo, cariño y ánimos durante todo el periodo académico.

Agradecer a mi tutor Javier Esparza Peidró, por integrarse, motivarme a superarme y ayudarme en el nuevo mundo de la programación a través de sus clases.

Tanto como a Andrea Carozzo, mi tutor de prácticas, por brindarme la oportunidad de profesionalizarme en el sector de la programación e inspirarme a .

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1 Introducción

La gestión de datos se ha vuelto una necesidad del día a día. Todo el mundo necesita tener un control organizado de sus emails, una lista de contactos con los que interactuar y hablar diariamente, o incluso una pila de tareas del día, semana, e incluso mes.

Esta gestión se vuelve más importante cuando los datos, archivos o contactos sobre los que necesitamos tener un firme control, podrían perjudicarnos legalmente al perderlos, olvidarnos de donde los dejamos por última vez, o incluso si los desechamos o borramos de nuestra computadora.

Hablamos de los contratos. Si en algún momento de tu vida has llegado a un acuerdo con otra persona mediante un contrato, lo más normal, después de haber firmado las dos o demás partes, es dejarlo estático en algún rincón de nuestro disco duro o perderlo por algún cajón o archivador de nuestra casa. Cuando realmente, este tipo de archivos deberían de ser fácilmente accesibles, inmutables y persistentes.

¿Por qué un gestor de contratos? De la misma forma que utilizamos gestores de correo como Gmail, Outlook, Yahoo, las personas que interactúan diariamente con los contratos deberían tener control total sobre ellos, una plataforma donde subir, gestionar, enviar, y mantener un registro de las acciones con sus contratos, con la ayuda de la blockchain.

La blockchain es una tecnología basada en el registro de transacciones, como si de un libro mayor se tratase. Como indica su nombre, las transacciones van almacenándose en bloques, cuando se llega al límite de información por bloque se genera uno nuevo a partir de este y quedan conectados, por tanto, el resultado final es una cadena de bloques conectados uno a uno. Algunas de sus ventajas más importantes son la inmutabilidad, persistencia, seguridad e incorruptibilidad.

Las ventajas de poder operar con un gestor de contratos con blockchain integrada es, que también podemos automatizar el registro de nuestros contratos hacia la blockchain gracias a los contratos inteligentes [1].

Los contratos inteligentes se tratan de “scripts”, código informático como si de un programa se tratase, esto quiere decir que los términos en un Smart Contract son puros comandos y líneas de código. Una de las grandes ventajas de estos se debe a su naturaleza, es un código visible por todos y que no se puede cambiar al existir sobre la tecnología blockchain. Como su nombre indica, el trabajo de los Smart Contracts es detectar ciertas acciones del usuario automáticamente y actuar acorde a lo programado. Por ejemplo, cuando un usuario acepte un contrato, el Smart Contract detectará esta acción y enviará una transacción a la blockchain guardando información relevante sobre el suceso.

1.1.1 ¿Por qué Blockchain?

Esta tecnología, conceptualizada por primera vez por la persona o grupo de personas, Satoshi Nakamoto, fue implementada por primera vez como uno de los componentes principales de la criptodivisa Bitcoin, donde la blockchain actúa como un libro de contabilidad público para todas las transacciones producidas dentro de la red de Bitcoin [2].

La finalidad de esta tecnología es alcanzar un consenso, construido y alcanzado por los propios miembros de la red. Para comprometer los datos, un atacante requeriría de una mayor potencia de cómputo y presencia en la red que el resultante de la suma de todos los restantes nodos combinados.

Por tanto, la tecnología blockchain es especialmente adecuada en el caso de que se necesite almacenar datos de forma progresiva en el tiempo, sin posibilidad de modificación, y distribuyendo la confianza entre todos los miembros de la red.

Para este proyecto, la integración de blockchain es un valor añadido, ya que, al trabajar con contratos, documentos altamente sensibles, comparten ciertas características clave con la tecnología blockchain:

- Consenso: Para que una transacción sea válida, todos los participantes deben de estar de acuerdo con su validez
- Procedencia: Los participantes saben de dónde viene el eslabón, quién participaba en él, y cómo ha cambiado su propietario con el tiempo
- Inmutabilidad: Ningún participante puede modificar la lista de registros una vez que se haya guardado en la cadena. Si una transacción es errónea, se debe hacer una nueva para corregirla, tras lo cual ambas transacciones serán visibles.

1.1.2 ¿Qué son los Smart Contracts?

Los contratos inteligentes surgieron en 1993 cuando el criptógrafo estadounidense Nick Szabo comenzó a usar el término. Szabo propuso convertir los contratos tradicionales a este sistema, pero fracasó debido a las limitaciones técnicas de la época. Pero con el nacimiento de Bitcoin en 2009, las cosas cambiaron. Los contratos inteligentes necesitaban un sistema de pago que pudiera ejecutarlos y ahora era posible.

Un contrato inteligente es un programa informático que facilita, protege y hace cumplir un acuerdo registrado entre dos o más partes (como individuos u organizaciones). Como tal, es útil para negociar y definir acuerdos que afectan ciertas acciones como resultado del cumplimiento de ciertas condiciones.

Un contrato inteligente es un programa que existe en un sistema que no está controlado por ninguna de las partes o sus agentes y que ejecuta contratos automáticos que actúan como declaraciones si-entonces (if-then) de otro programa informático. Excepto que esto se hace de una manera que interactúa con activos reales. Cuando se activan condiciones preprogramadas que no están sujetas a evaluación humana, el contrato inteligente ejecutará los términos del contrato apropiados.

1.2 Motivaciones

Siempre hemos trabajado con aplicaciones web que nos faciliten nuestro día a día. Calendarios que nos recuerdan eventos importantes, redes sociales que nos informan de las novedades del día, son ejemplos de servicios sobre los que dependemos diariamente. Pero, y que sucede con aplicaciones que más allá de facilitarte la vida, ¿te sean una verdadera herramienta en la que puedas descargar cierta parte de tus responsabilidades?

Aquí nace la idea de Kontractum, una aplicación web donde puedas crear y firmar contratos con el valor añadido de subir las firmas a la blockchain. Para lograr una correcta funcionalidad de un gestor de contratos debemos tener en cuenta ciertos aspectos:

- Los contratos deben ser irremplazables, por ende, únicos.
- El usuario debe poder guardar y almacenar, dependiendo del tipo, estado o fecha de creación, sus contratos.
- El usuario debe poder filtrar fácilmente entre sus contratos.
- Los contratos en la aplicación deben ser inmutables, seguros, persistentes y fácilmente accesibles.
- Debe existir transparencia total entre las dos partes contratantes.

El objetivo principal de la aplicación es poder ofrecer valor mediante el servicio de gestión de contratos, y que pueda ser fácilmente integrable a la rutina de los usuarios, como revisar el correo por las mañanas o leer las noticias mientras te haces un café.

1.3 Estado del arte

Actualmente, la tecnología blockchain se encuentra en pañales, se trata de una tecnología emergente y es normal que las grandes empresas tarden en analizar y estudiar si es conveniente incorporarla en sus proyectos o servicios.

Bien es cierto que, dentro del mundo blockchain, muchas empresas nacen ya preconcebidas como blockchain-nativas, es decir, ya incorporan o tienen pensado incorporar esta tecnología desde un inicio. Al existir, también, múltiples cadenas de bloques en el ecosistema, muchas empresas se especializan alrededor de diferentes blockchain, las más importantes, Bitcoin, Ethereum, Cardano, Binance Smart Chain, Polkadot… Cada una contiene su propia blockchain y su propio conjunto de empresas o start-ups que construyen sus servicios con tales blockchain como base.

En cuanto a grandes empresas que estén incorporando blockchain a sus servicios, nos podemos encontrar con que distintos sectores, como el automovilístico, logístico y tecnológico, estarían altamente interesados en incorporar esta tecnología.

Por la parte automovilística nos encontraríamos a la FORD, y TOYOTA, que planean implementar esta tecnología para mejorar los sistemas de movilidad, así como de conducción autónoma.

En el caso de las logísticas, tendríamos a ALIBABA y FedEX, donde buscarían rastrear productos de lujo en sus plataformas de comercio electrónico y buscar una mejor alternativa para la resolución de disputas entre clientes.

Por último, entre las empresas tecnológicas, tenemos a gigantes como Apple, Facebook, Samsung o Google, entre los cuales, sus principales motivos para integrar la cadena de bloques son, mejorar la seguridad de datos en la nube, mejorar la privacidad de los datos de sus usuarios e incluso rumores de que Apple podría estar planeando patentar su propia blockchain para manejar datos de marca temporal.

En el siguiente apartado, realizaremos un análisis sobre algunas de las soluciones ya existentes en el mercado, que tienen cierta similitud, en cuanto a funcionamiento, a la aplicación que queremos crear.

1.3.1 Entorno

Para la búsqueda de aplicaciones se han tenido en cuenta ciertas características que todas las plataformas deben de compartir para asemejarse a nuestro proyecto.

Tras realizar una búsqueda y análisis a los actuales productos más relevantes del mercado en cuanto al ámbito de la gestión de contratos, hemos encontrado los siguientes:

1.3.1.1 PandaDoc

PandaDoc agiliza la gestión de contratos desde la venta hasta la ejecución y renovación del contrato. Incorpora funciones como la colaboración, control de versiones, historial de documentos y firma electrónica.

Dirección web: https://www.pandadoc.com/

Figura 1 - Landing page de PandaDoc

1.3.1.2 Juro

Juro es una plataforma de flujo de trabajo por contrato de extremo a extremo diseñada para los negocios modernos. Ayuda a los equipos legales a realizar contratos de forma más eficiente. El producto habilitado para IA de Juro ofrece herramientas de creación, negociación, firma electrónica y análisis de contratos.

Dirección web: https://juro.com/

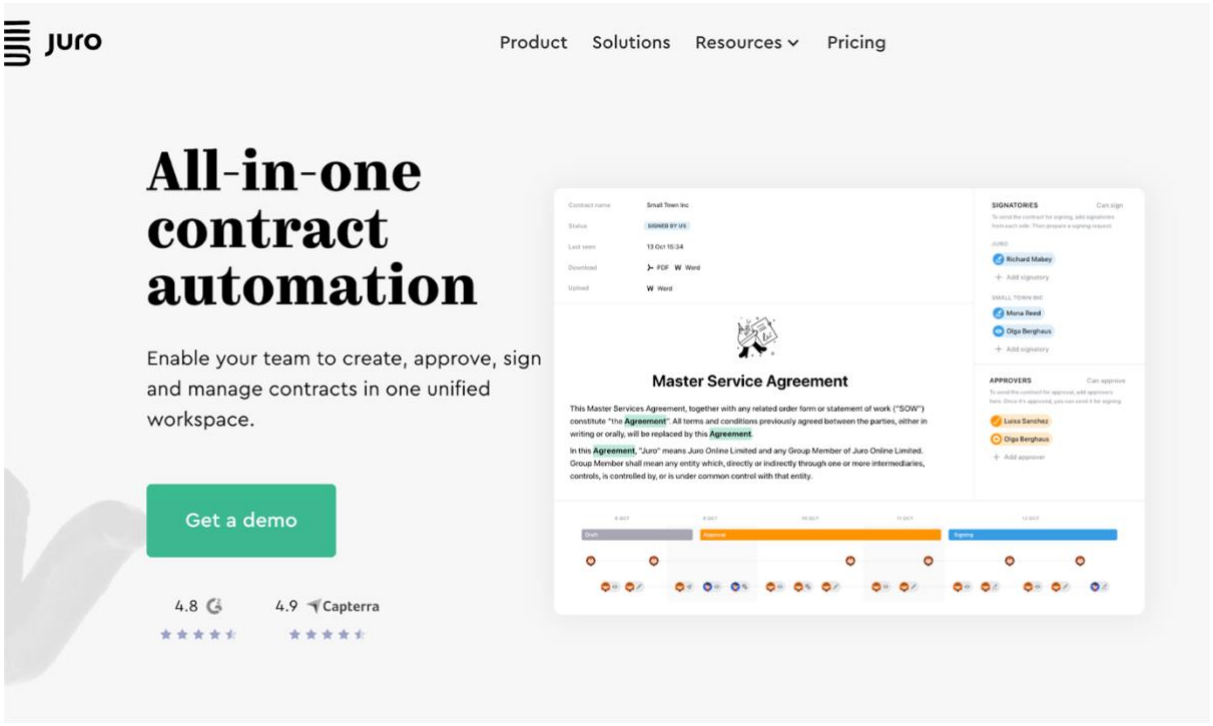

Figura 2 - Landing page de Juro

1.3.1.3 2Agree

2Agree es una solución SaaS para la gestión y el seguimiento eficaces de los contratos en todos los departamentos comerciales. El software de gestión de contratos de 2Agree proporciona a los administradores un control total de los contratos dentro de su organización. La aplicación tiene una función de administración de versiones incorporada que puede enviar notificaciones automáticamente cuando se necesita tomar una acción.

Dirección web: https://www.2agree.com/

Figura 3 - Landing page de 2Agree

1.3.1.4 ContractBook

Contractbook es una solución de gestión de contratos para empresas modernas. Crea, firma y almacena todos tus documentos legales en una plataforma digital para aumentar la transparencia, garantizar el cumplimiento y ahorrar tiempo. También ofrece una plataforma orientada al cliente donde los profesionales legales pueden supervisar y administrar los contratos de sus clientes de forma digital durante todo su ciclo de vida. La plataforma permite una estrecha comunicación, así como nuevas fuentes de ingresos y modelos de negocio.

Dirección web: https://contractbook.com/

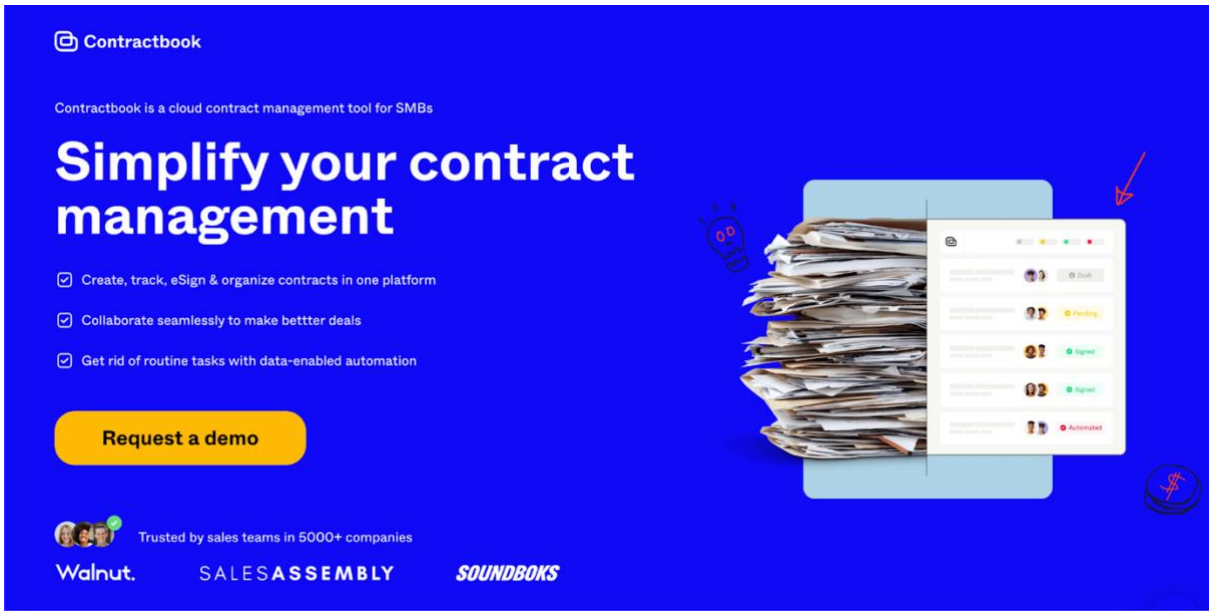

Figura 4 - Landing page de ContractBook

1.4 Síntesis del entorno

Tras realizar un estudio de mercado, analizando las aplicaciones similares, se ha detectado algunas características que comparten todas las webs, como son el registro de usuarios, cuotas mensuales o informes y estadísticas adaptadas a cada usuario. También cabe destacar que solo una de las webs analizadas tiene versión para teléfonos inteligentes.

En la siguiente tabla, sintetizamos las características que hemos identificado como importantes, para así tener una referencia o ver puntos débiles donde podría mejorar nuestra aplicación.

	PandaDoc	Juro	2Agree	ContractBook
Registro Clientes	Si	Si	Si	Si
Diseño web adaptativo	Si	Si	Regular	Si
Demo gratuita	Si	No	Si	No
Precio mensual	19,00$	49,00$	35,00$	330,00$
Aplicación móvil	Si	No	No	No
Cloud	Si	Si	Si	Si
Recordatorios	Si	Si	No	Si
Informes y estadísticas	Si	No	No	No

Tabla 1 - Comparativa entre plataformas

Tras este análisis, podemos apreciar una clara ventaja de PandaDoc sobre el resto de plataformas, ofreciendo un servicio bastante amplio a un precio muy competitivo si lo comparamos con el resto.

Por lo tanto, buscamos hacerle frente a PandaDoc ofreciendo las mismas e incluso mejores funcionalidades.

1.5 Objetivos

Se desea diseñar una plataforma digital de gestión de contratos que cubra las siguientes necesidades:

- Gestión y clasificación de contratos.
- Intermediación entre las distintas partes.
- Facilidad de uso implementando una GUI.
- Flujos similares a las de un gestor de email.
- Mantener registros de las firmas almacenándolas en la blockchain.

El objetivo principal del proyecto es servir a usuarios una plataforma donde subir y administrar sus contratos personales, brindando una interfaz sencilla e intuitiva. A su vez brindar características a los contratos, tales como: estados, fecha de creación, acciones dependiendo de estado...

Además de incorporar Web 3.0 a la plataforma, lo que nos permite la integración de contratos inteligentes y la comunicación con la blockchain. Otros objetivos a desarrollar en un futuro, persiguiendo la evolución de la aplicación, sería lograr desarrollar una aplicación móvil para atraer a más nicho de mercado, crear una lista de contactos entre usuarios o introducir plantillas de contratos por defecto.

La meta del sistema es llegar a ser un modelo de negocio rentable, obtener beneficios por los servicios prestados y convertirse en una de las aplicaciones de gestión de contratos más utilizadas.

1.6 Metodologías

1.6.1 Kanban

El desarrollo del proyecto ha sido marcado siguiendo la metodología KanBan.

Por contextualizar, KanBan [3] fué desarrollado por Taiichi Ohno, ingeniero japonés de Toyota a finales de la época de los 40. De hecho, “Kanban” es una combinación de dos palabras japonesas: 看 (Kàn), que significa “signo” o “señal visual”, y 板 (Bǎn), que significa “tablero”.

KanBan emplea tableros, llamados tableros KanBan, que son una forma visual de gestionar tu proyecto en diferentes etapas o estados, son muy populares entre equipos de ingeniería de productos y desarrollo software.

Con esta metodología, encontramos un equilibrio entre las tareas que se deben realizar y la disponibilidad de cada miembro del equipo. En el caso de este proyecto de final de carrera, el equipo está formado únicamente por una persona, el alumno, por lo que esta metodología se centrará en la

mejora continua, donde las tareas a realizar se “extraen” de una tabla de acciones pendientes en un flujo de trabajo constante.

La metodología se implementa a través de los anteriormente mencionados, tableros KanBan. Cada columna representa una etapa de trabajo. De forma simple, podríamos dividir el trabajo en los siguientes tableros: Trabajo pendiente, tareas en progreso y tareas terminadas.

Las tareas individuales se posicionarían en su tablero correspondiente y, dependiendo de su orden en la lista, estarían ordenadas dependiendo de cuán importante sea cada tarea.

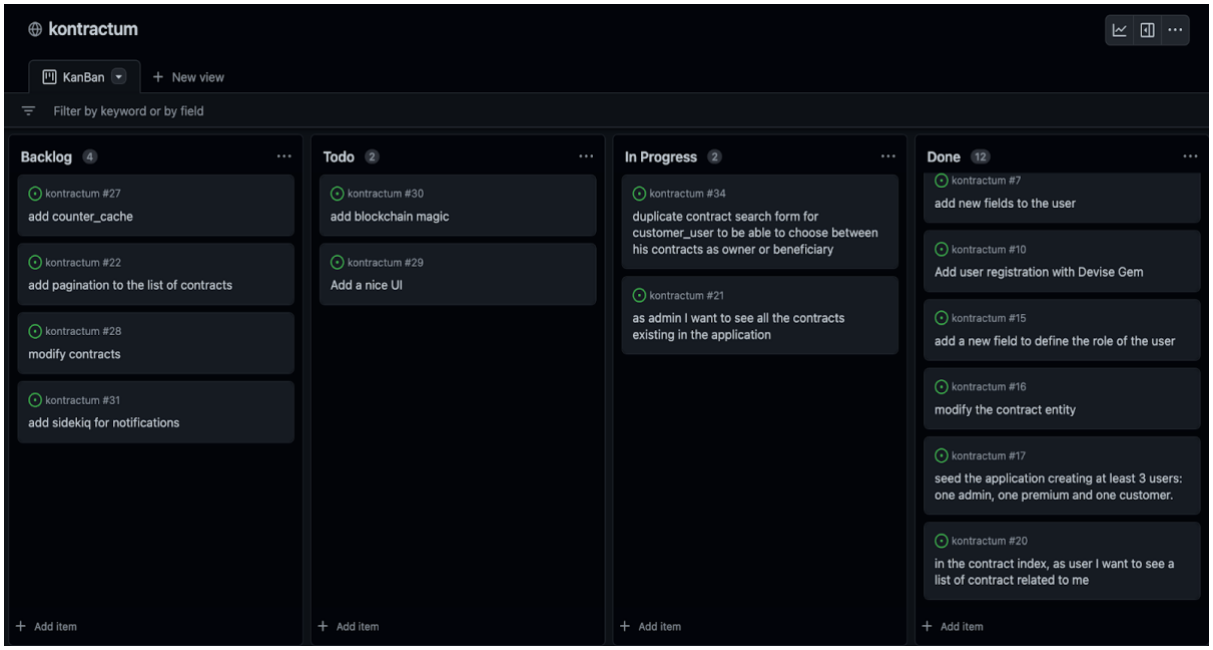

Figura 5 - Metodología KanBan

1.6.2 Test Driven Development

TDD [4] son las siglas de Test Driven Development. Es un proceso de desarrollo que consiste en codificar pruebas, desarrollar y refactorizar de forma continua el código construido.

La idea principal de esta metodología es realizar de forma inicial las pruebas unitarias para el código que tenemos que implementar. Es decir, primero codificamos la prueba y, posteriormente, se desarrolla la lógica de negocio.

Es una visión algo simplificada de lo que supone, ya que bajo mi punto de vista también nos aporta una visión más amplia de lo que vamos a desarrollar.

Este proceso consta de tres fases:

- Fallar los tests
- Pasar los tests
- Refactorizar

Se trata de codificación y refactorización, escogemos un requisito fácilmente implementable, codificamos la prueba para cumplir el requisito, verificamos que la prueba falla, codificamos la implementación para que la prueba funcione, ejecutamos las pruebas correspondientes y refactorizamos.

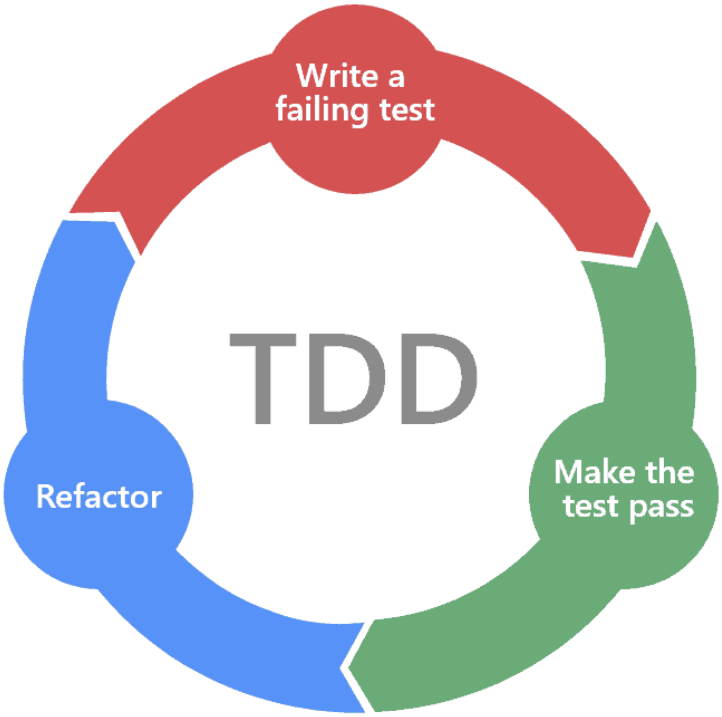

Figura 6 - Metodología TDD

1.6.3 Diagrama de Gantt

A continuación, se muestra el diagrama de Gantt [5], en el cual, hemos de remarcar dos cosas, en primera instancia los fines de semana no están dentro de nuestro calendario.

Lo segundo es que no se han incluido los recursos, puesto que el proyecto lo ha llevado a cabo una sola persona y, por lo tanto, en cada uno de los puntos los recursos para esa persona alcanzarán el 100% y no habrá recursos sobre asignados.

Por contextualizar, la realización del proyecto se ha realizado en 3 fases, las cuales podríamos agrupar como:

- Fase Inicial
- Fase Funcional
- Fase Final

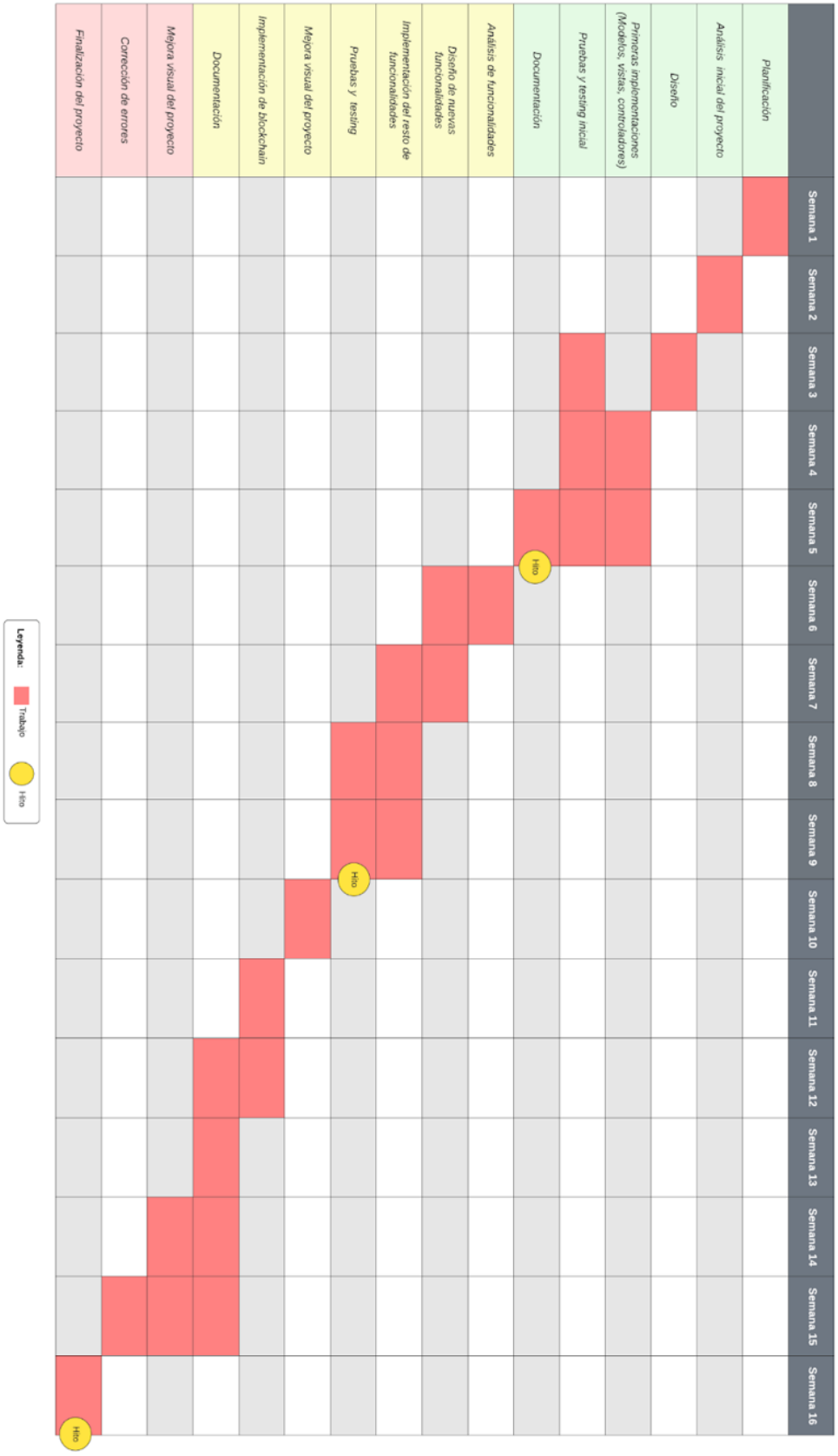

Figura 7 - Diagrama de Gantt

1.7 Alcance

Este proyecto va destinado a aquellos usuarios que requieran un cierto nivel en cuanto a gestión de sus contratos, ya sea porque necesitan una mayor centralización de estos para tener un mayor control y seguridad sobre ellos, o porque tienen muchos tipos de contratos de diferentes temáticas y necesitan un espacio donde poder almacenarlos de forma organizada y ordenada.

1.8 Impacto Esperado

Con la realización de este proyecto, se busca añadir valor y funcionalidad, a una parte de la gestión de archivos que nunca se había valorado mejorar.

De hecho, este proyecto va un paso más allá e integra blockchain, permitiendo que los usuarios tengan un acercamiento inicial al operar con esta nueva tecnología de forma totalmente guiada y autónoma.

1.9 Estructura de la obra

Parte 1 - Introducción

La parte principal de la documentación se centra en presentar de forma introductoria las motivaciones y objetivos del proyecto y realizar un breve análisis del entorno antes de presentar y definir los problemas a solucionar o el valor que se pretende aportar al mercado.

Parte 2 - Definición del problema

Listaremos con detalle los requisitos y funcionalidades de la aplicación, mostrando de forma gráfica y lógica los requisitos, así como los casos de uso e incluso diagramas de procesos que expliquen los pasos lógicos a tomar.

Parte 3 - Solución del problema

Se describirá el cómo se ha construido la aplicación, explicando a través de diagramas UML, dibujos y esquemas la arquitectura de la misma, qué tecnologías se han utilizado, y de que capas o componentes está formada la misma.

Parte 4 - Resultado

Se llevará a cabo un pequeño tour por las funcionalidades de la aplicación, mostrando capturas de pantalla e incluso algún trailer de la aplicación

Parte 5 - Conclusiones

Se explicará qué es lo que se ha logrado con este TFG, alguna idea de mejora o alguna parte que no se ha podido implementar.

Parte 6 - Bibliografía

Todas las referencias que se han empleado en el proyecto.

2 Definición del problema

Los siguientes puntos tratarán de identificar y explicar los diferentes tipos de requisitos, funcionales y no funcionales detectados en nuestra plataforma, así como las dependencias necesarias para poder llevar a cabo la acción.

También se abordarán los casos de uso desde un punto de vista gráfico

2.1 Casos de uso

En UML, el diagrama de casos de uso permite definir las funcionalidades del sistema. Para ello utiliza una notación gráfica.

En estas notaciones mostramos las distintas funcionalidades que nos proporciona la aplicación.

Observaremos distintos casos de uso, dependiendo del rol asignado al usuario, podrá tener más, o menos acceso a funcionalidades de la aplicación.

Recordar que, los distintos roles, listados de mayor a menor acceso a funciones, serían los siguientes:

- Admin
- Premium Customer
- Customer
- Banned
- Blocked

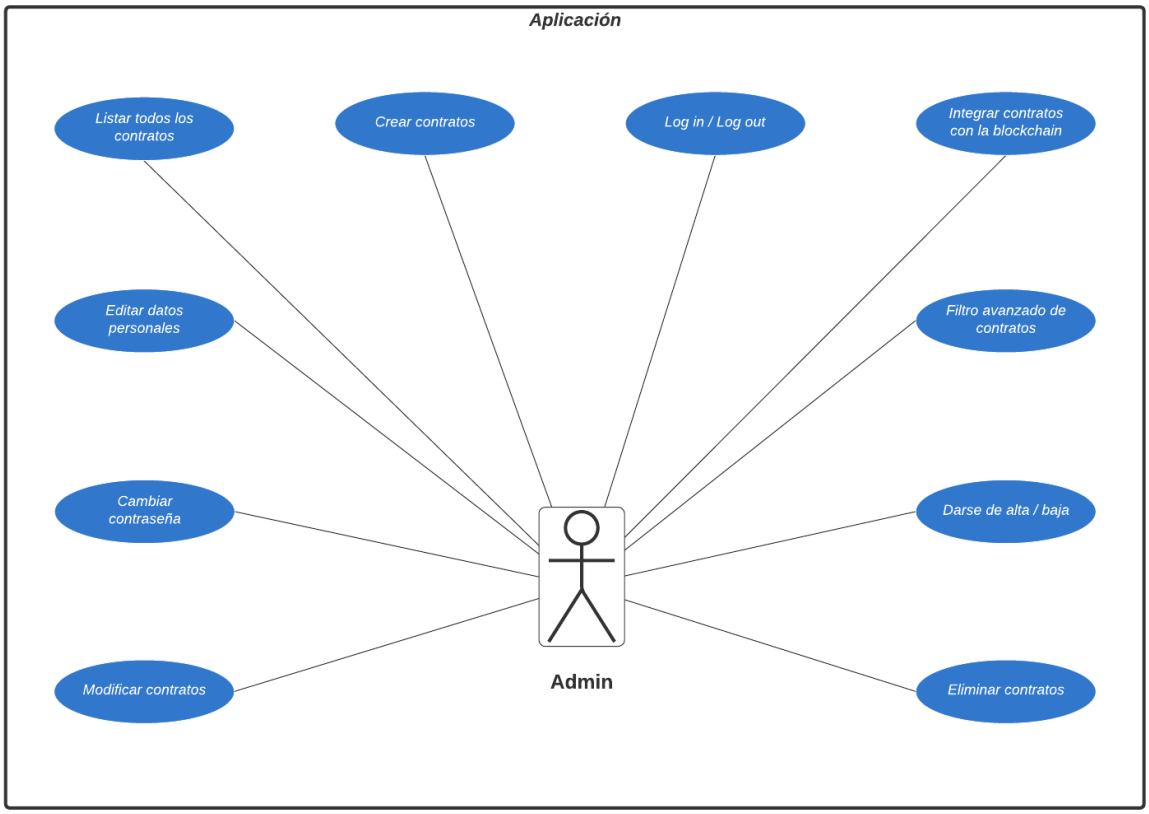

Figura 8 - Diagrama casos de uso. Admin

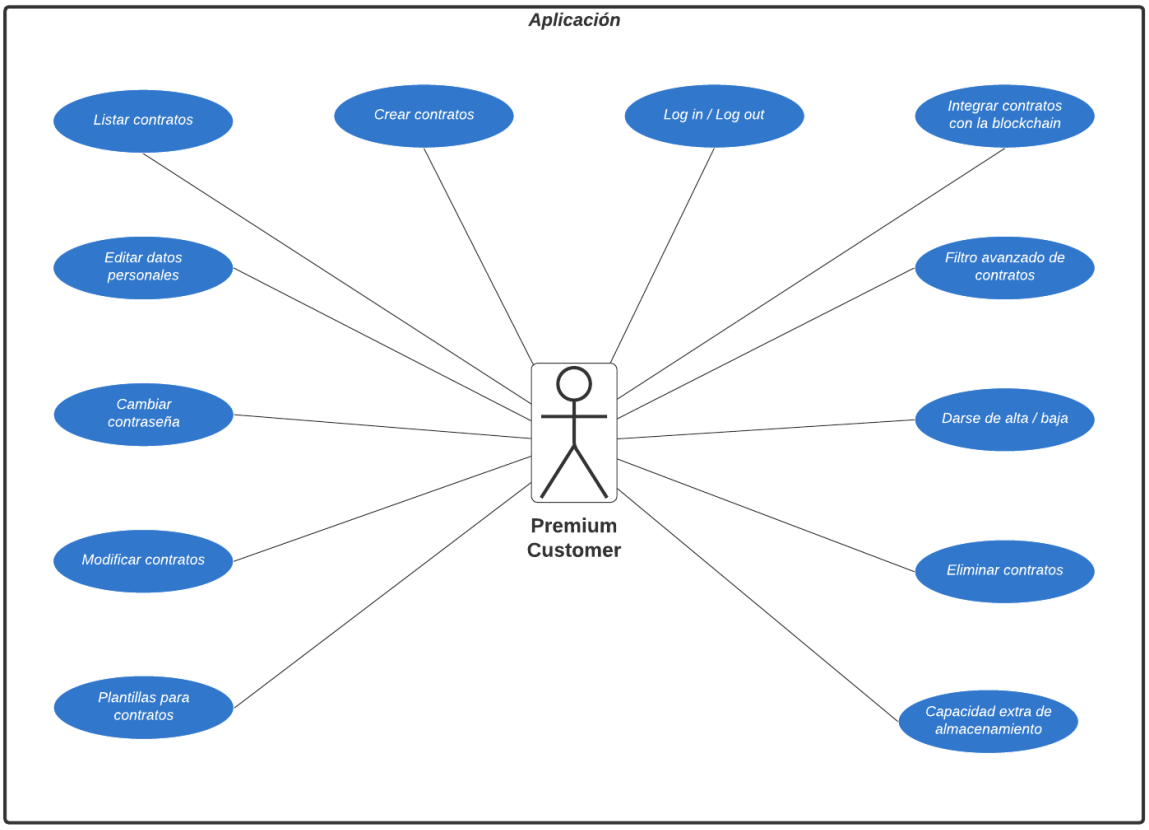

Figura 9 - Diagrama casos de uso. Premium customer

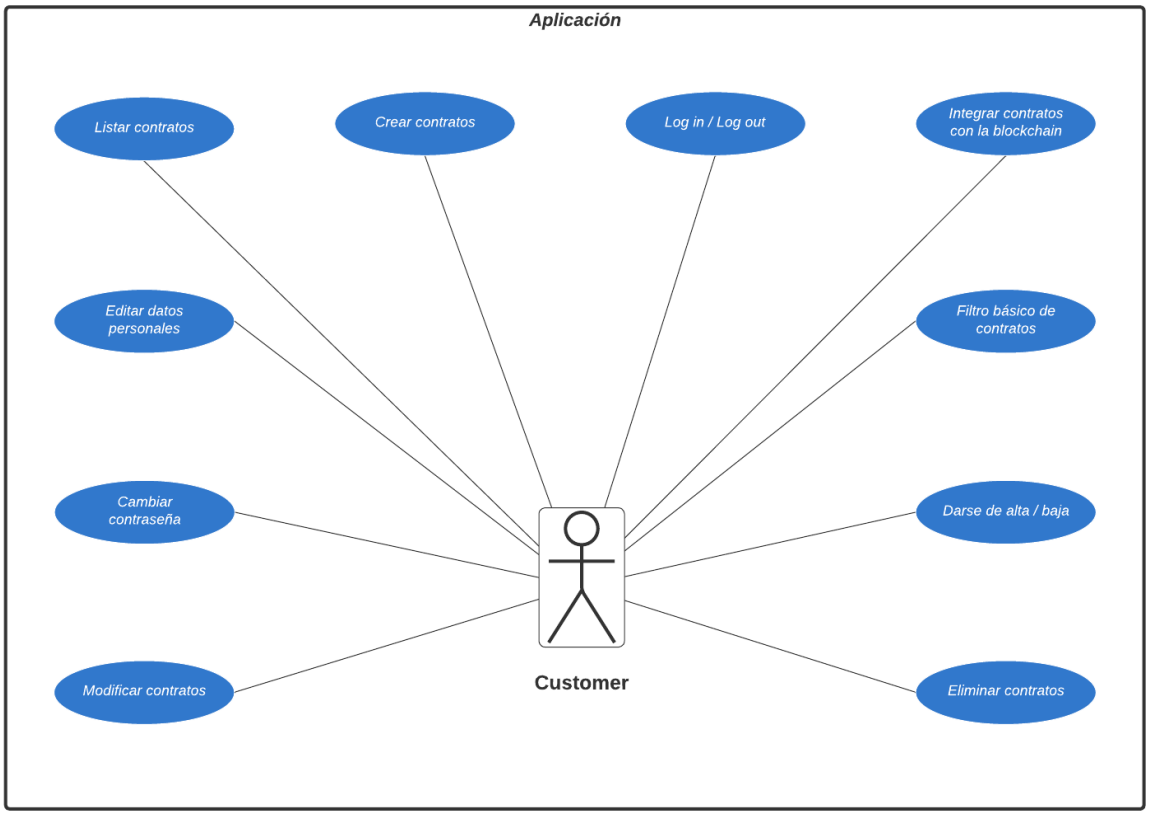

Figura 10 - Diagrama casos de uso. Customer

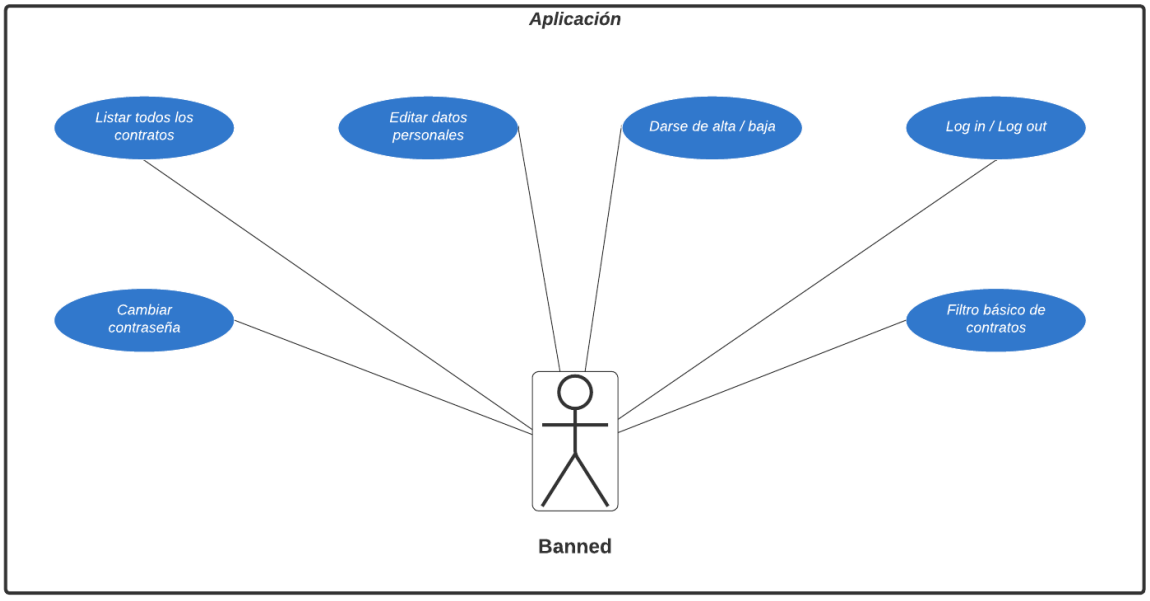

Figura 11 - Diagrama casos de uso. Banned

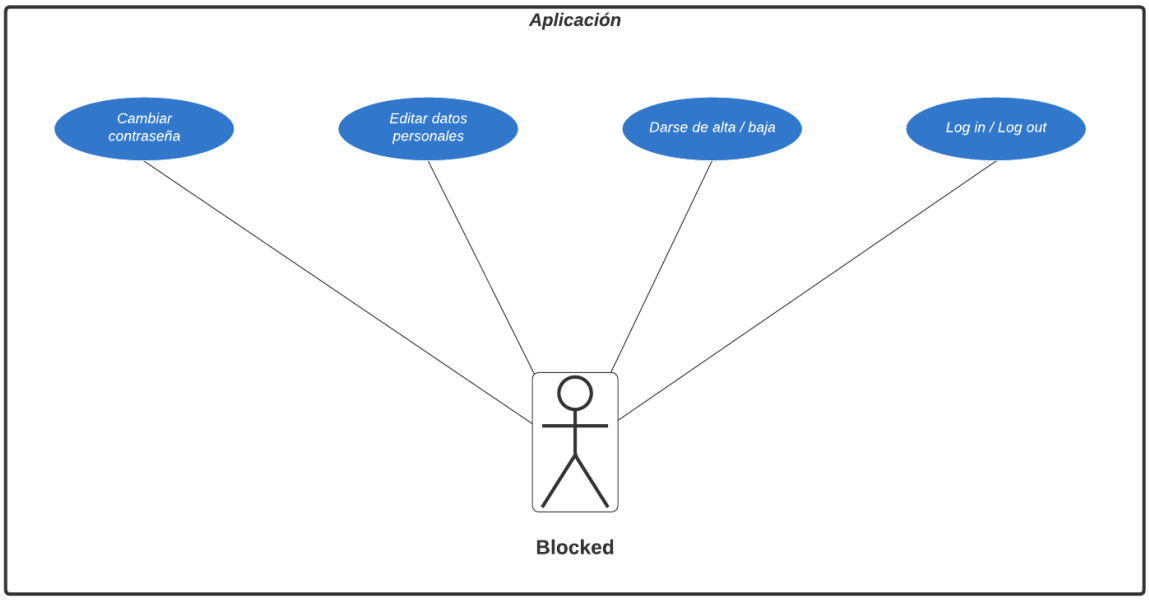

Figura 12 - Diagrama casos de uso. Blocked

2.2 Análisis

2.2.1 Análisis de requisitos funcionales

Un requisito funcional puede ser definido como una breve descripción de lo que un sistema debería poder hacer. La aplicación debe de ser capaz de realizar estos requisitos para poder cumplir su funcionalidad.

RF-001	**Dar de alta a usuario**
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	La aplicación deberá de poder dar de alta a usuarios con sus datos correctamente validados

Tabla 2 - Requisito funcional: Dar de alta usuario

RF-002	**Recuperar contraseña usuario**
Autor	Fco Javier Gozalvo Cervera

Dependencias	RF-001 [Dar de alta a usuario]
Descripción	La aplicación deberá facilitar via email una recuperación de la contraseña

Tabla 3 - Requisito funcional: Recuperar contraseña usuario

RF-003	**Entrar a aplicación**
Autor	Fco Javier Gozalvo Cervera
Dependencias	RF-001 [Dar de alta a usuario]
Descripción	La aplicación deberá permitir el acceso a sus funciones a aquellos usuarios registrados

Tabla 4 - Requisito funcional: Entrar a aplicación

RF-004	**Dar de baja a usuario**
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	Si el usuario lo requiere, la aplicación deberá de borrar todos sus datos

Tabla 5 - Requisito funcional: Dar de baja a usuario

RF-005	**Crear contratos**
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	La aplicación permitirá a usuarios, si su rol se lo permite, crear contratos

Tabla 6 - Requisito funcional: Crear contratos

RF-006	**Editar contrato**
Autor	Fco Javier Gozalvo Cervera
Dependencias	RF-005 [Crear contratos]
Descripción	La aplicación permitirá editar los contratos siempre y cuando el usuario sea su creador

Tabla 7 - Requisito funcional: Editar contrato

RF-007	Eliminar contratos
Autor	Fco Javier Gozalvo Cervera
Dependencias	RF-005 [Crear contratos]
Descripción	La aplicación permitirá eliminar contratos si el usuario es su creador

Tabla 8 - Requisito funcional: Eliminar contratos

RF-008	Listar contratos
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	La aplicación listará todos aquellos contratos donde el usuario sea creador o sea beneficiario.

Tabla 9 - Requisito funcional: Listar contratos

RF-009	Editar perfil
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	La aplicación permitirá editar la información personal del usuario

Tabla 10 - Requisito funcional: Editar perfil

RF-010	Editar contraseña
Autor	Fco Javier Gozalvo Cervera
Dependencias	-
Descripción	La aplicación permitirá editar la contraseña de acceso al usuario

Tabla 11 - Requisito funcional: Editar contraseña

2.2.2 Análisis de requisitos no funcionales

Un requisito no funcional, en referencia a ingeniería de software, un requisito que detalla y especifica criterios que pueden ser relevantes en el momento de juzgar las operaciones en vez del comportamiento específico.

Los requisitos no funcionales se encuentran listados a continuación:

RNF-001	Navegador Web
Autor	Fco Javier Gozalvo Cervera
Descripción	La aplicación deberá funcionar en los navegadores más utilizados del mercado

Tabla 12 - Requisito no funcional: Navegador web

RNF-002	Diseño responsive
Autor	Fco Javier Gozalvo Cervera
Descripción	La aplicación deberá adaptarse a las dimensiones de pantalla en las que se esté visualizando

Tabla 13 - Requisito no funcional: Diseño responsive

RNF-003	Aprendizaje
Autor	Fco Javier Gozalvo Cervera
Descripción	La aplicación deberá permitir que un usuario promedio sea capaz de manejar las funcionalidades de la WebApp

Tabla 14 - Requisito no funcional: Aprendizaje

2.2.3 Modelos de la aplicación

El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

M-001	Usuario
Autor	Fco Javier Gozalvo Cervera
Datos específicos	• ID del usuario • Nombre • Apellidos • Número de identidad • Tipo de documento identidad • Email • Fecha nacimiento • Dirección • Código Postal • Ciudad • Rol en el sistema

	• Fecha de creación • Fecha de actualización • Contraseña encriptada

Tabla 15 - Requisito informativo: Usuario

M-002	**Contratos**
Autor	Fco Javier Gozalvo Cervera
Datos específicos	• Título • Descripción • Tipo • Estado • Fecha creación • Fecha actualización • Propietario (usuario) • Beneficiario (usuario) • Pagabilidad

Tabla 16 - Requisito informativo: Contratos

2.3 Procesos

Para entender mejor los diferentes procesos de la aplicación, los modelaremos utilizando diagramas de flujo.

Los diagramas de flujo representan la secuencia lógica o los pasos que tenemos que dar para realizar una tarea mediante unos símbolos, dentro de ellos se describen los pasos a realizar.

Un diagrama de flujo debe proporcionar una información clara, ordenada y concisa de todos los pasos a seguir.

Por lo dicho anteriormente, podríamos decir que: "Un diagrama de flujo es una representación gráfica o simbólica de un proceso".

En nuestro caso, se han creado 3 diagramas de flujo para tener una visión más concreta sobre cada proceso principal. Separaremos los tres diagramas en tres grupos:

- Proceso inicial
- Proceso sobre datos personales
- Proceso sobre contratos

2.3.1 Proceso inicial

En el proceso inicial se representa la secuencia lógica para crear una cuenta e iniciar sesión en la aplicación, así como los pasos a seguir si el usuario no recuerda la contraseña.

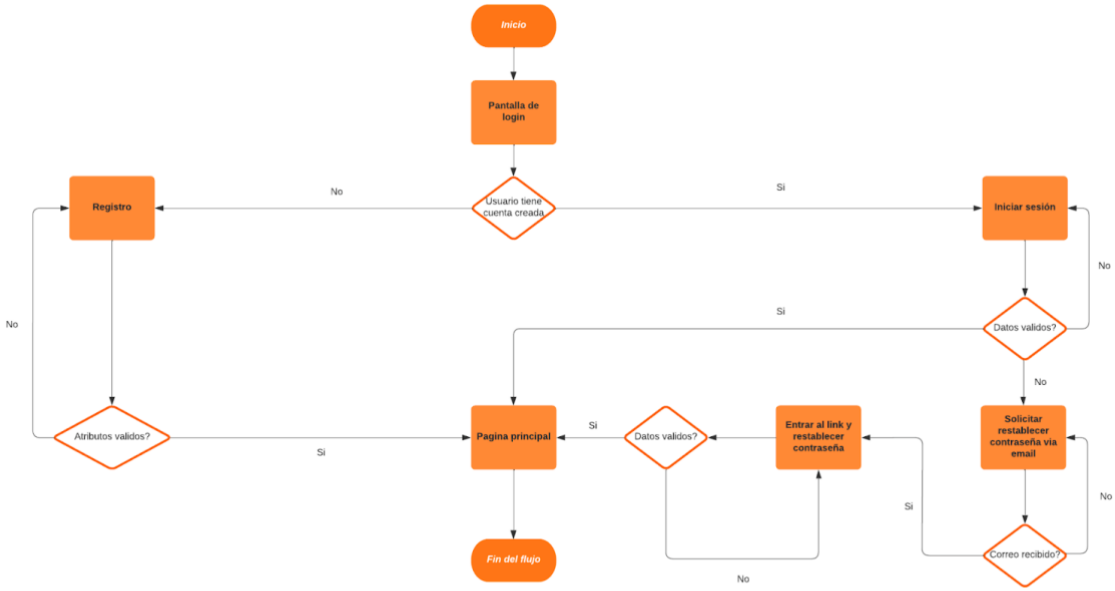

Figura 13 - Proceso inicial

2.3.2 Proceso sobre datos personales

En el siguiente proceso, representaremos los pasos a seguir si el usuario quisiera efectuar acciones tales como; editar su información personal, cambiar su contraseña o darse de baja del sistema.

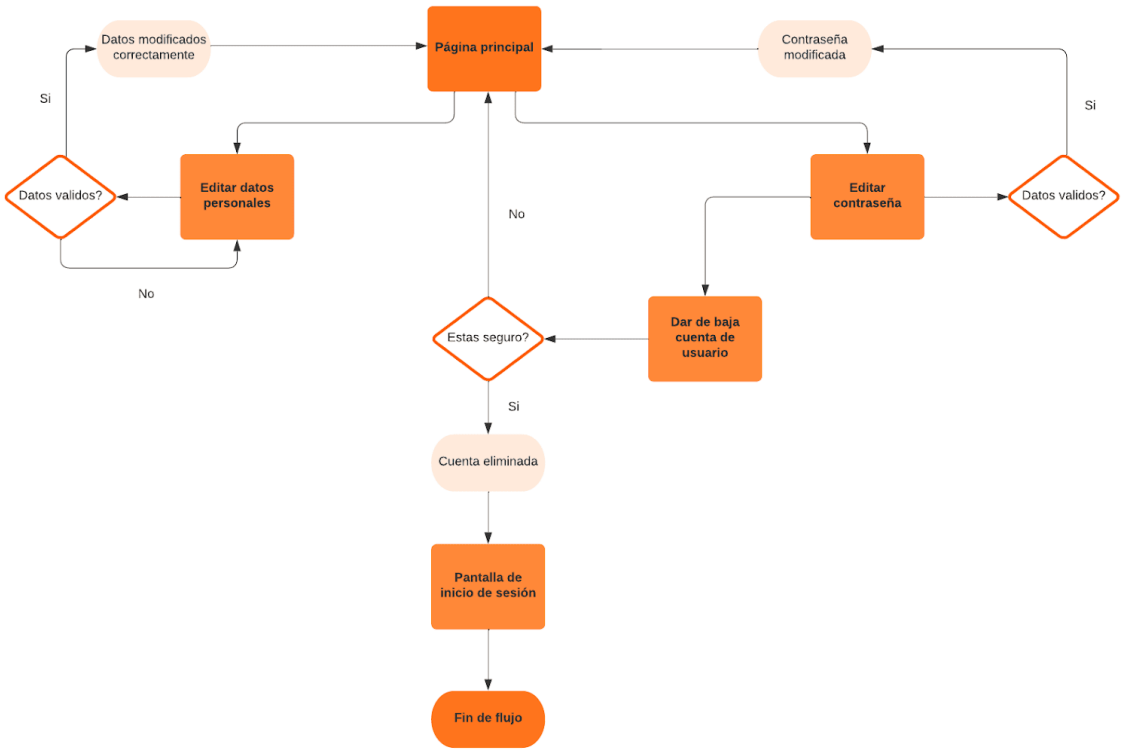

Figura 14 - Proceso sobre datos personales

2.3.3 Flujo del estado de los contratos

Los contratos contienen un atributo llamado Estado para poder identificar las diferentes fases sobre las cuales puede pasar un contrato. Entre ellos, existen 3 estados finales, Archived, Approved y Tokenized.

*Los estados token (tokenization_requested, tokenizable, tokenized) están relacionados con la interacción contrato – blockchain, donde requieren acciones vía metamask, software que posibilita transacciones a la blockchain.

Los posibles estados son los siguientes:

- Proposed: Es el estado default al crear un contrato, este estado da a entender que el contrato ha sido propuesto y se espera una respuesta por parte del beneficiario. Un contrato que se encuentre en este estado, podrá ser aceptado o rechazado.

- Approved: Es el estado que obtiene el contrato si, tras su propuesta, es aceptado. Este estado puede considerarse como estado final. A un contrato aprobado se le puede pedir una modificación o una tokenización*.

- Modification_requested: El contrato pasa al estado Modification_requested cuando el beneficiario le pide al creador un cambio sobre el contrato. El creador puede, o no, aceptar estos cambios.

- Modification_in_progress: Si el creador del contrato acepta la propuesta de modificación por parte del beneficiario, el estado del contrato pasa a Modification_in_progress.

- Edited: Una vez el beneficiario finaliza la modificación del contrato, el contrato editado es enviado al creador y el estado del contrato pasa a Edited. Las posibles acciones por parte del creador son aceptar los cambios realizados, o rechazarlos. En caso de aceptar los cambios, el contrato pasaría a Approved, en caso de rechazarlos, a Archived.

- Archived: En este estado entrarían los contratos "descartados", ya sean rechazados desde un inicio por el beneficiario, modificaciones no aceptadas o denegadas por el creador. Se considera un estado final, ya que una vez el contrato se encuentra en este estado, únicamente puede ser borrado por su creador.

- Tokenization_requested: El creador del estado, una vez este se encuentre en estado Approved, puede pedir al beneficiario una tokenización del estado, es decir, encriptar su información y subirla a la blockchain para dejar huella del acuerdo.

- Tokenizable: Si el beneficiario acepta la propuesta de tokenización por parte del creador, el estado del contrato pasaría a Tokenizable, en caso de no aceptar, el contrato se mantendría Approved.

- Tokenized: Mientras el contrato se encuentre en el estado tokenizable, ambas partes deben firmar el contrato. Un usuario, al firmar, realiza una llamada a la función crear contrato del Smart Contract, esta función acepta como parámetros dos strings, el hash del contrato y el correo del usuario. Por tanto, al firmar se enviaría una transacción con estos dos parámetros. Una vez ambas firmas estén presentes en la blockchain, el estado del contrato pasa a ser Tokenized. El acuerdo entre ambas partes ya se encuentra en la blockchain, y el creador puede borrar, o no, el contrato de la plataforma.

Figura 15 – Flujo de estado de los contratos

En este esquema podemos ver a simple vista las acciones de cada usuario dependiendo de su rol y del estado en el que se encuentre el contrato.

Acciones en contratos dependiendo de rol y estado	Archived	Approved	Proposed
Owner / Creador	Eliminar	Eliminar Preguntar tokenización	Editar contrato Eliminar contrato
Beneficiary / Beneficiario	--	Preguntar modificación	Aceptar contrato Rechazar contrato

Acciones en contratos dependiendo de rol y estado	Modification_requested	Modification_in_progress	Edited
Owner / Creador	Aceptar prop. modificación Rechazar prop. modificación	--	Aceptar modificación Rechazar modificación
Beneficiary / Beneficiario	--	Editar contrato Finalizar modificación	--

Acciones en contratos dependiendo de rol y estado	tokenization_requested	tokenizable	tokenized
Owner / Creador	--	Firmar contrato	Eliminar contrato
Beneficiary / Beneficiario	Preguntar modificación Aceptar tokenización Rechazar tokenización	Firmar contrato	--

Figura 16 - Resumen acciones sobre contratos

3 Solución del problema

3.1 Arquitectura del sistema

La arquitectura del software se refiere a la estructura y las relaciones entre las diferentes partes del software y sus propiedades visibles externas.

Su objetivo principal es dotar a un sistema de gestión de datos de una determinada calidad basada en el rendimiento, el ahorro de tiempo, la disponibilidad y facilidad de uso, así como la capacidad de cambiar el sistema para adaptarse a nuevas necesidades y atributos.

Nuestra aplicación cuenta con una arquitectura en la que los clientes (front-end) solicitan unos servicios al servidor (back-end), y este se los proporciona (cliente-servidor). La forma que tenemos de enviar o recibir datos, los cuales serán almacenados o consultados en una base de datos relacional orientado a objetos, se basa en hacer una petición POST al servidor y GET en caso de querer recibirlos.

3.1.1 Arquitectura Cliente-Servidor

La arquitectura cliente-servidor [6] se refiere al sistema que aloja, proporciona y administra la mayoría de los recursos y servicios solicitados por los clientes. En este modelo, todas las solicitudes y servicios se envían a través de una red, conocida como red cliente-servidor.

- Cliente: Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito, una vez que son servidas sus solicitudes, termina el trabajo.
- Servidor: Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. Al ser posible implantarlo como aplicaciones de programas, puede ejecutarse en cualquier sistema donde exista TCP/IP y junto con otros programas de aplicación. El servidor comienza su ejecución antes de comenzar la interacción con el cliente.

Figura 17 - Diagrama Cliente-Servidor

3.1.2 Ventajas y desventajas de la arquitectura Cliente-Servidor

La gran mayoría de aplicaciones que utilizamos en nuestro día a día utiliza este tipo de arquitectura. Las ventajas de este modelo respecto de otras posibles arquitecturas de red serían:

- Centralización
- Fácil mantenimiento
- Escalabilidad, podemos aumentar la capacidad de clientes y servidores por separado.

Por otro lado, la mayor desventaja es que a mayor número de clientes, más problemas para el servidor, debido a la congestión en el tráfico que se genera, además de los problemas que habría si se dieran ataques como DDOS (Distributed denial of service), caídas del servidor que dejarían a sus usuarios colgados, o posibles estafas a través de phishing o ataques MITM (Man in the middle).

El cliente no ejecuta ninguna lógica de negocio ni procesa ningún tipo de datos. Todo este trabajo recae sobre el servidor, por tanto, nuestra arquitectura se define como server-rendering.

3.2 Tecnologías utilizadas

3.2.1 Ruby on Rails

La tecnología predominante en el proyecto es el conocido framework Ruby on Rails [7], programado en el lenguaje orientado a objetos, Ruby. Este framework engloba tecnologías como HTML (más específicamente, HTML Embedded Ruby), CSS, Javascript y Ruby, también incluye un ORM [8], un Object-relational Mapping, mecanismo que hace posible direccionar, acceder y manipular objetos sin tener que considerar cómo estos objetos se relacionan con sus fuentes de datos, llamado Active Record.

ActiveRecord [9] es un patrón de persistencia que facilita la creación y el uso de objetos de negocios cuyos datos requieren almacenamiento persistente en una base de datos, es una implementación del patrón ActiveRecord descrito por Martin Fowler, que a su vez es una descripción de un ORM, el cual nos otorga diferentes funcionalidades, como:

- Representar modelos y sus datos
- Representar asociaciones entre estos modelos
- Representar jerarquías de herencia entre relaciones de modelos
- Validar modelos antes de quedar persistentes en la BBDD.
- Realizar operaciones de bases de datos al estilo orientación a objetos

Cabe destacar también DataMapper, otro ORM para Rails, se trata de una librería o gema, ya que por defecto el ORM nativo de Rails es ActiveRecord. Fue desarrollada para solucionar ciertas deficiencias detectadas en ActiveRecord.

Escoger entre un ORM u otro para nuestra aplicación ha sido tarea fácil, debido a sus diferencias y para qué está diseñado cada uno. Para nuestra aplicación, al ser CRUD-based y tener varias relaciones entre nuestros modelos, ActiveRecord es la solución perfecta.

En definitiva, es un framework muy completo, que facilita mucho la vida al programador y que permite también, entre otras cosas:

- Crear nuestras propias API-Rest para nuestras aplicaciones
- Sintaxis simple, amigable y con buena legibilidad
- Utiliza patrón MVC
- Puede soportar múltiples bases de datos
- Pensado para agilidad, desarrollo rápido y el ahorro de tiempo al evitar arquitecturas complejas

Figura 18 - Icono de Ruby on Rails

3.2.2 HTML5

La tecnología utilizada para mostrar las vistas al usuario y que este interactúa con la aplicación será HTML [10].

Es el lenguaje más importante y conocido para la creación de páginas web, dándoles una estructura y un contenido. Basado en etiquetas, breves instrucciones de apertura y cierre.

Figura 19 - Icono de HTML5

3.2.3 Ruby

El lenguaje base de Ruby on Rails, es Ruby [11]. Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Su elegante sintaxis se siente natural al leerla y fácil al escribirla. Es un lenguaje interpretado y orientado a objetos.

Figura 20 - Icono de Ruby

3.2.4 CSS y Bootstrap

El lenguaje de estilo elegido para maquetar y crear una web más atractiva al usuario es CSS [12], junto a Bootstrap [13], framework para frontend de CSS, diseñado para facilitar el proceso de desarrollo de los sitios web responsivos y orientados a los dispositivos móviles, proporcionando una colección de sintaxis para diseños de plantillas.

Figura 21 - Icono de CSS

Figura 22 - Icono de Bootstrap

3.2.5 Javascript

Un lenguaje menos predominante en el proyecto, pero, que ha sido utilizado para la incorporación e integración de la blockchain en la aplicación, ha sido Javascript [14].

Javascript es un lenguaje de programación orientado a objetos empleado principalmente para la creación de web dinámicas del lado del cliente y en ocasiones del lado del servidor.

Figura 23 - Icono de Javascript

3.2.6 Solidity

Otro lenguaje incorporado al proyecto para poder integrar blockchain y en especial, contratos inteligentes a la aplicación, es el lenguaje enfocado a la programación de Smart Contracts, Solidity [15]. Este lenguaje nos permite crear contratos inteligentes y desplegar estos en varias plataformas blockchain, como por ejemplo Ethereum [16].

Figura 24 - Icono de Solidity

3.2.7 Ganache

Ganache [17] es una cadena de bloques personal para el desarrollo rápido de dApps [18] de Ethereum y Corda. Ganache permite desarrollar, implementar y probar las dApps en un entorno seguro y determinista. Ganache UI es una aplicación de escritorio que admite la tecnología Ethereum

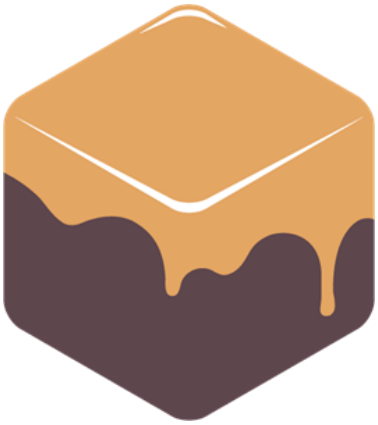

Figura 25 - Icono de Ganache

3.2.8 Metamask

MetaMask [19] es un plugin para el navegador que sirve como wallet de Ethereum, y se instala como cualquier otro plugin normal. Una vez instalado, permite a los usuarios almacenar Ether y otros tokens ERC-20, permitiéndoles realizar transacciones a cualquier dirección de Ethereum.

Figura 26 - Icono de Metamask

3.2.9 SQLite3

El sistema de gestión de base de datos integrado a Ruby on Rails por defecto [20] es SQLite3. Este gestor de base de datos relacional, es compatible con las características ACID y está contenida en una biblioteca relativamente pequeña, permite capacidades de hasta 2 terabytes de información. Para este proyecto de final de carrera, se ha elegido este gestor de bases de datos, ya que no esperamos una escalabilidad enorme en los primeros pasos de la aplicación, pero en el caso de tener que ampliar tamaño de la base de datos, Ruby on Rails permite la integración de SQL, MongoDB o MariaDB, entre otros.

Figura 27 - Icono de SQLite3

3.3 Servidor

Para diseñar la arquitectura del servidor se ha seguido el patrón Modelo Vista Controlador [21].

Un patrón, en el ámbito del desarrollo de software, puede definirse como una solución o grupo de soluciones que ya han sido probadas antes para problemas recurrentes en ciertos contextos. En otras palabras, un patrón es la forma o método con el que se han podido solucionar problemas en el desarrollo de aplicaciones, aprovechando la experiencia de las corporaciones y en general, de los desarrolladores.

Los patrones de diseño experimentaron un boom en la década de los 90 a partir de la publicación del libro Pattern Designs [22] escrito por un grupo de informáticos conocidos como Gang of Four, en el que se recopilaban cerca de 23 patrones de diseño.

Entre los patrones creacionales explicados en el libro, el patrón MVC (Modelo – Vista - Controlador) fue uno de los más adaptados por los frameworks modernos, capaz de separar los datos y la lógica de negocio entre la interfaz de usuario y el módulo encargado de gestionar comunicaciones y eventos.

3.3.1 Patrón Modelo - Vista - Controlador

Utilizaremos este patrón para poder organizar, de forma lógica, la estructura del código dentro del servidor.

El patrón Modelo Vista Controlador, separa la parte gráfica de una aplicación, de los procesos lógicos y de los datos de la misma.

Está constituido por las siguientes partes:

- La vista: Es la Interfaz gráfica, es con lo que el usuario de la aplicación interactua y es donde se reciben las “órdenes” y se presentan los resultados de los procesos al usuario.

- El controlador: Es el modificador del modelo, es decir, es el que modifica los valores de las variables, objetos, datos en general, lo hace de acuerdo a lo solicitado por el usuario a través de la interfaz gráfica (vista).

- El modelo: Es la representación específica de la información con la que se está operando en el instante de la ejecución de la aplicación, representa las variables, objetos, datos en general que se están modificando de acuerdo a lo que el usuario solicite.

Cuando la aplicación requiere del almacenamiento de datos, regularmente se utiliza una base de datos. La información almacenada en esa base de datos depende de los procesos desarrollados en el modelo. Cuando no existe base de datos se emplean archivos para el almacenamiento de la información, dependiendo de la implementación, dichos archivos pueden ser incluidos dentro del modelo.

El uso del patrón MVC, específicamente en este proyecto, impone una separación de responsabilidades entre los diferentes componentes de nuestra aplicación, por lo que hace más claro la misión u objetivo de cada capa. Además, sigue la convención de “Convention Over Configuration”, como consecuencia, obtenemos un desarrollo muy organizado y una mayor velocidad de desarrollo en equipo.

3.3.2 Flujo del patrón MVC

Podemos encontrar diferentes implementaciones del modelo MVC, pero el flujo de procesos que se sigue usualmente es el siguiente:

1. El usuario interactúa de alguna forma con la vista, por ejemplo, haciendo click en un enlace, o enviando un formulario
2. El controlador recibe, a través de los objetos de la interfaz, la notificación de la acción solicitada por el usuario, y gestiona el evento que llega a través de un handler.
3. El controlador accede al modelo, posiblemente, actualizandolo acorde a la acción realizada por el usuario, en el caso de haber rellenado un formulario, guardando o actualizando la información que el usuario ha escrito, en la base de datos.
4. El controlador delega a la vista el deber de desplegar la interfaz de usuario, habiendo obtenido la vista los datos del modelo para generar la interfaz apropiada, donde ya se reflejan los cambios que el usuario ha producido anteriormente sobre el modelo. El modelo no tiene, o al menos no debería, tener conocimiento directo sobre la vista.
5. La interfaz de usuario espera nuevas acciones por parte del usuario para repetir el proceso.

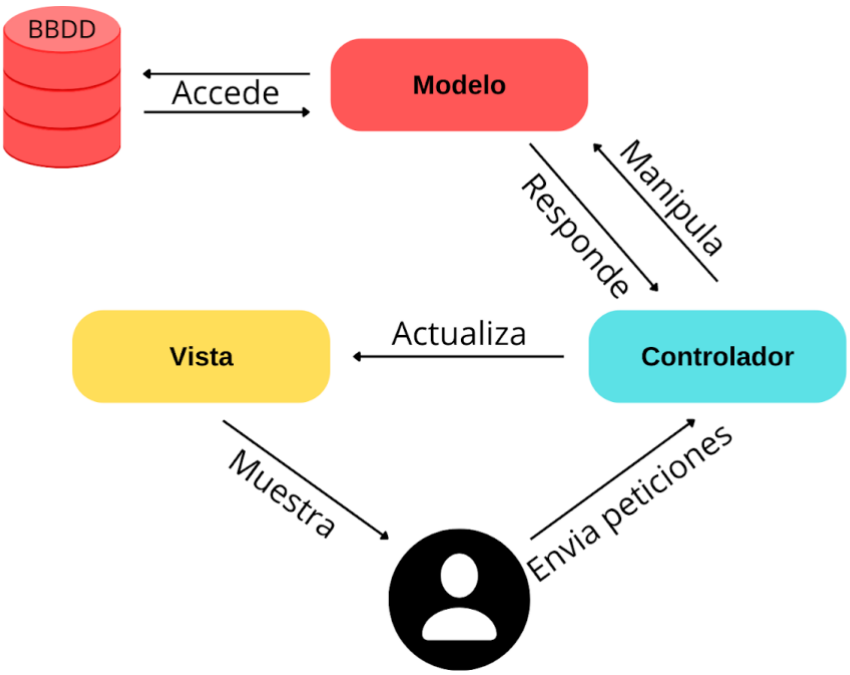

Figura 28 - Diagrama patrón MVC

3.3.3 Modelo

En la capa Modelo encontraremos siempre una representación de los datos del dominio, es decir, aquellas entidades que nos servirán para almacenar información del sistema que estamos desarrollando. En nuestro caso, en el modelo existirán las siguientes clases: Usuario y Contrato.
El modelo es responsable de:

- Acceder a la capa de almacenamiento de datos. Importante separar el modelo del sistema de almacenamiento
- Definir las reglas de negocio, por ejemplo, “Un usuario no puede registrarse si tiene más de 100 años” o “Un contrato no puede crearse si no incluye título”
- Incluir mecanismos de persistencia, por tanto, encargado de gestionar el almacenamiento y recuperación de datos.

Los modelos creados en nuestra aplicación son los siguientes:

MOD-001	**Usuario**
Autor	Fco Javier Gozalvo Cervera
Datos específicos	• ID del usuario • Nombre • Apellidos • Número de identidad • Tipo de documento identidad • Email • Fecha nacimiento • Dirección • Código Postal • Ciudad • Rol en el sistema • Fecha de creación • Fecha de actualización • Contraseña encriptada

Tabla 17 - Modelo Usuario

MOD-002	**Contratos**
Autor	Fco Javier Gozalvo Cervera
Datos específicos	• Título • Descripción • Tipo • Estado • Fecha creación • Fecha actualización • Propietario (usuario) • Beneficiario (usuario) • Pagabilidad

Tabla 18 - Modelo Contrato

También se mostrará como las entidades del proyecto se relacionan entre sí mediante un diagrama de Entidad-Relación.

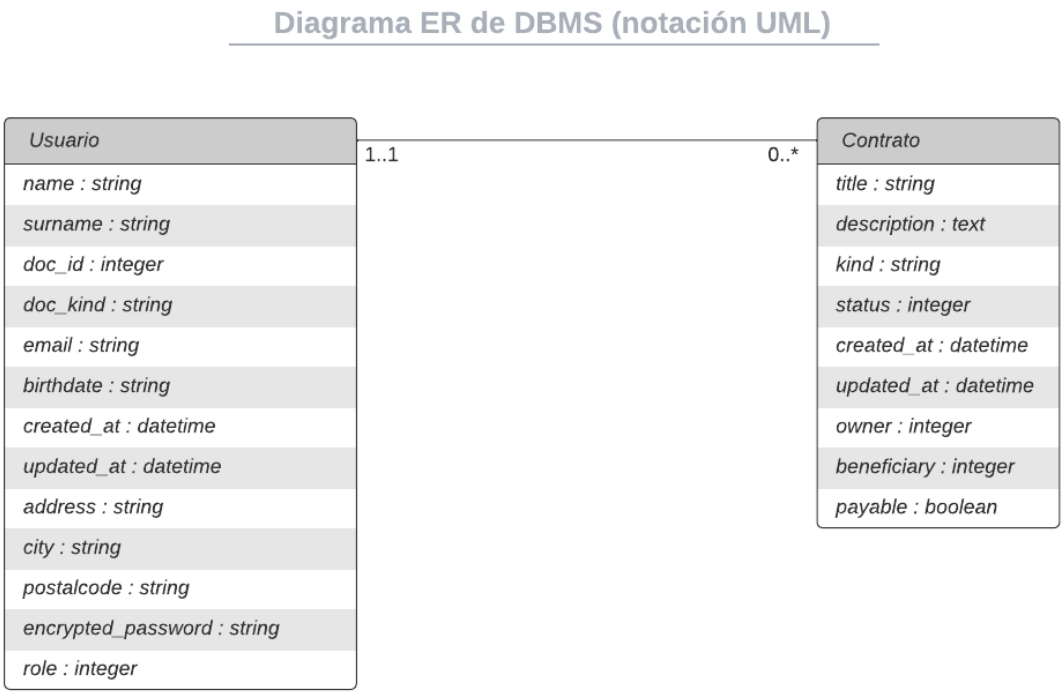

Figura 29 - Diagrama Entidad-Relación

3.3.4 Controladores

Esta capa se ocupa de comunicarse con el modelo para almacenar o recuperar datos y con la capa de presentación o vista para atender las peticiones y presentar los resultados. El controlador es responsable de:

- Actuar como intermediario entre usuario y sistema, así como un coordinador general del sistema.
- Recibir y gestionar los eventos de entrada (un clic en un determinado botón, un envío de formulario)
- Contener reglas de eventos, por ejemplo, “Si ocurre evento A, entonces se ejecuta acción B”. Estas acciones pueden ser peticiones, tanto al modelo como a las vistas. Una de estas peticiones puede ser una llamada a algún método o función.

Podemos observar el flujo que va a seguir un usuario no registrado para darse de alta en la web.

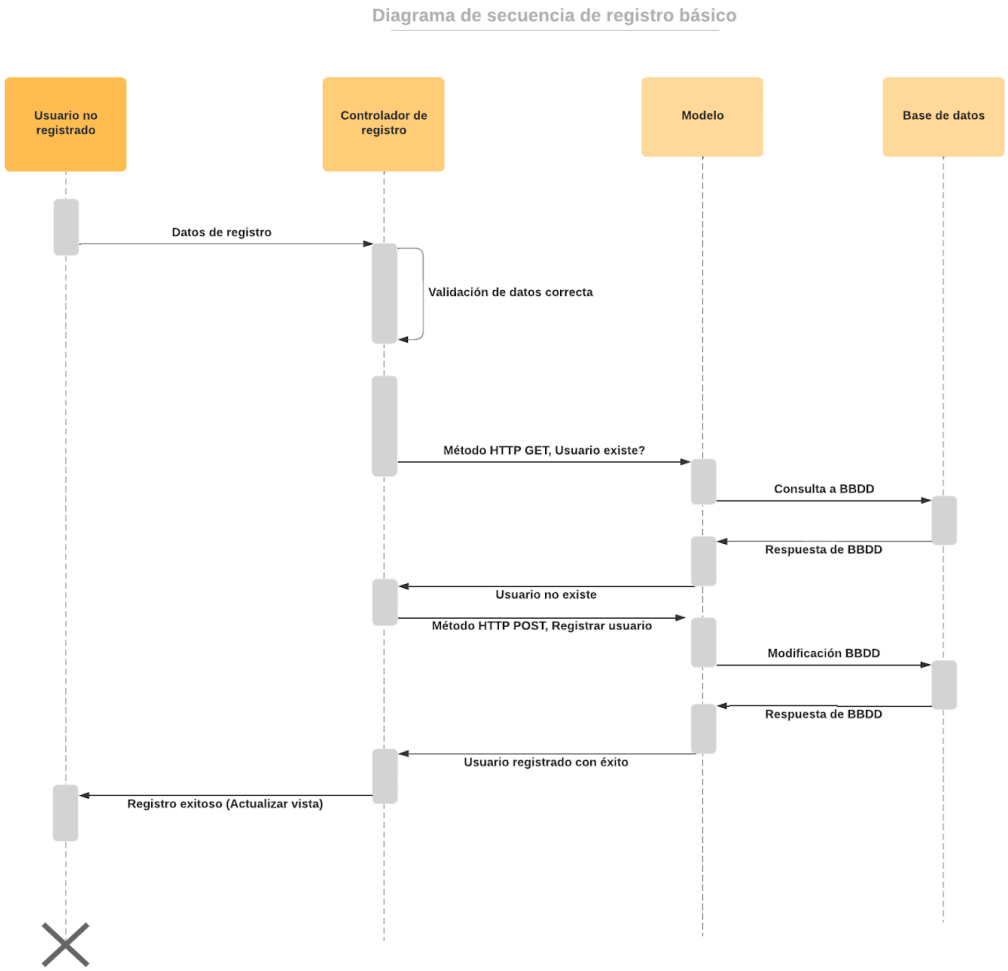

Figura 30 - Diagrama de secuencia de registro básico

El proyecto consta de 5 controladores para las diferentes vistas y los modelos que les conciernen.
Se han configurado los siguientes controladores:

- Application_controller: Es el controlador principal de la aplicación. Es la clase de la que todos los demás controladores heredarán. Una buena práctica es no llenarlo mucho de código y mantenerlo limpio.

- Users_controller: Es el controlador del modelo Users. Contiene todas las funciones CRUD, así como los parámetros requeridos para realizar operaciones en las diferentes funciones.

- Contracts_controller: Es el controlador del modelo Contracts. Contiene todas las funciones CRUD, así como los parámetros requeridos para realizar operaciones en las diferentes funciones.

- Profiles_controller: Es el controlador dedicado para el update de usuarios.

- Home_controller: Es el controlador dedicado para la vista principal de la aplicación. En este controlador se encuentran funciones de filtrado para la gema Ransack, así como funciones de paginación.

3.3.5 Vistas

Los componentes de la Vista son los responsables de generar la interfaz de nuestra aplicación, es decir, de componer las pantallas, páginas, o cualquier tipo de resultado utilizable por el usuario o cliente del sistema. De hecho, suele decirse que la vista es una representación del estado del modelo en un momento concreto y en el contexto de una acción determinada.

Las vistas serán las responsables de:

- Contener los elementos de interacción, que permitan al usuario enviar información e invocar acciones en el sistema

- Recibir datos del modelo y mostrarlos al usuario

- Tener un controlador asociado (normalmente instanciado)

Para realizar esquemas de las vistas de la aplicación se ha utilizado el software online Figma. Esta página nos permite crear wireframes y un sencillo flujo de vistas para esquematizar su futuro funcionamiento.

El diseño ha sido llevado a cabo siguiendo ciertos patrones, tanto la elección de los colores del UI como la distribución de los elementos se han realizado después de realizar un análisis sobre psicología del color y patrones sobre diseño web. Se ha realizado aplicando el enfoque de Diseño Centrado en el Usuario (DCU) [23] para el desarrollo de la interfaz gráfica de usuario de la aplicación.

Los colores principales escogidos varían entre un gris apagado #282828 y un blanco roto para el background de la página #EEEEEE. La selección de estos colores tiene como finalidad generar en el usuario durabilidad, importancia y duración por parte del gris oscuro, como ligereza y minimalismo por parte del blanco roto.

La distribución de los elementos principales de la página se ha hecho en 2 columnas principales. La primera columna, que contiene acciones de creación de contrato, botón de Inbox y filtros de estado de contratos, como recibidos y enviados. La segunda columna recibe más atención al ser casi tres cuartos de pantalla, una tabla contiene el listado de contratos creados, al estilo Gmail. Cada fila de la tabla representa un contrato y contiene:

- El título resaltado para poder acceder a la información del contrato
- Breve descripción del contrato para mejor identificación
- Fecha de creación del contrato
- En el caso de ser el propietario del contrato, aparecerán botones de borrar y editar
- En el caso de ser beneficiario, opciones de aceptar o rechazar contrato

Ahora se mostrarán algunos de los wireframes realizados:

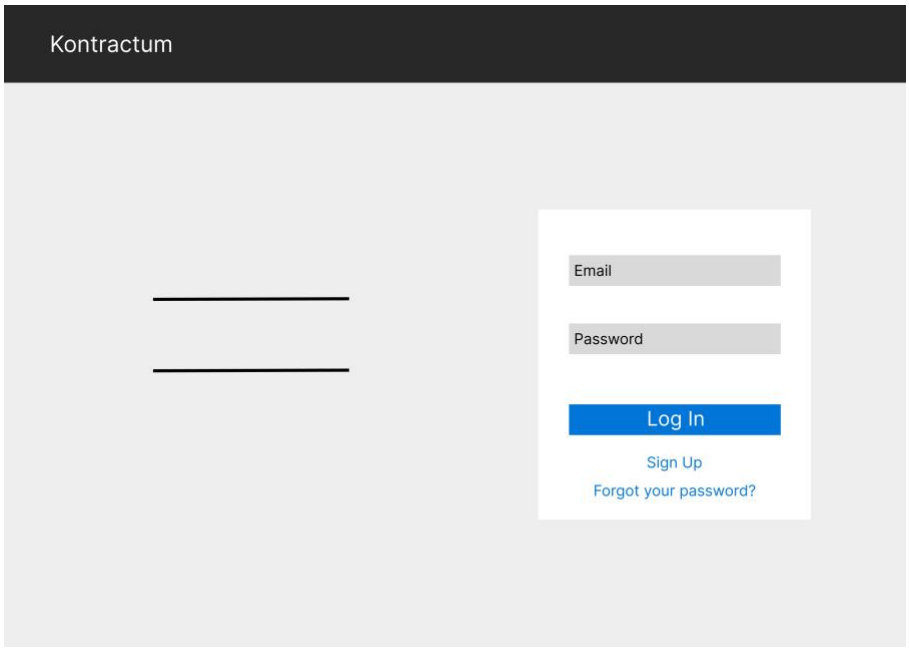

Figura 31 - Wireframe del Login

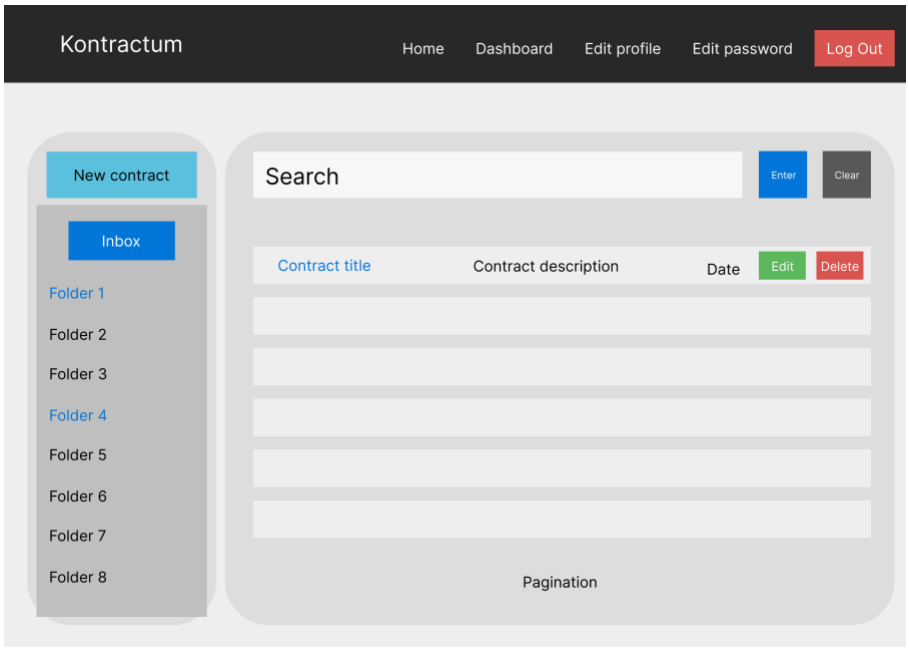

Figura 32 - Wireframe de la pantalla principal

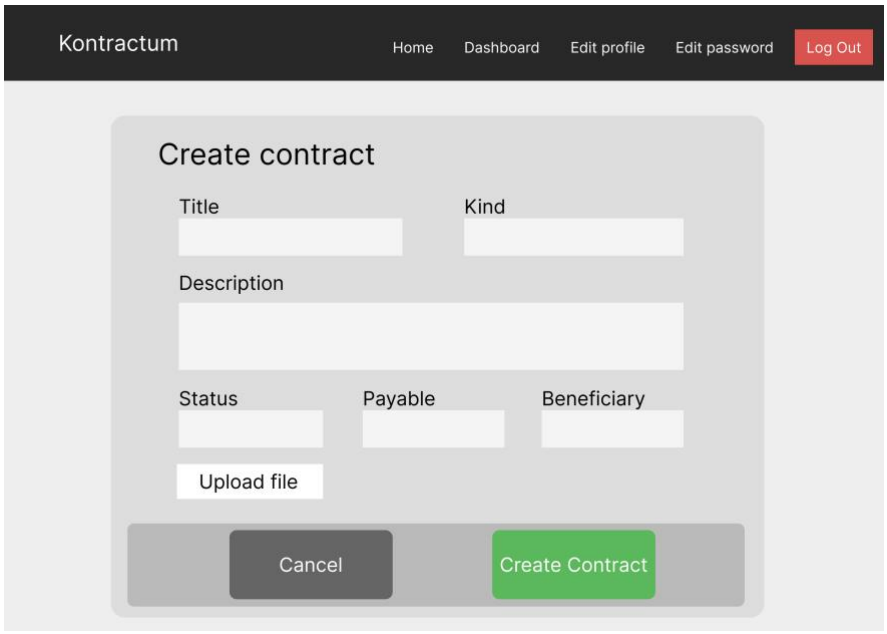

Figura 33 - Wireframe de la vista Crear Contrato

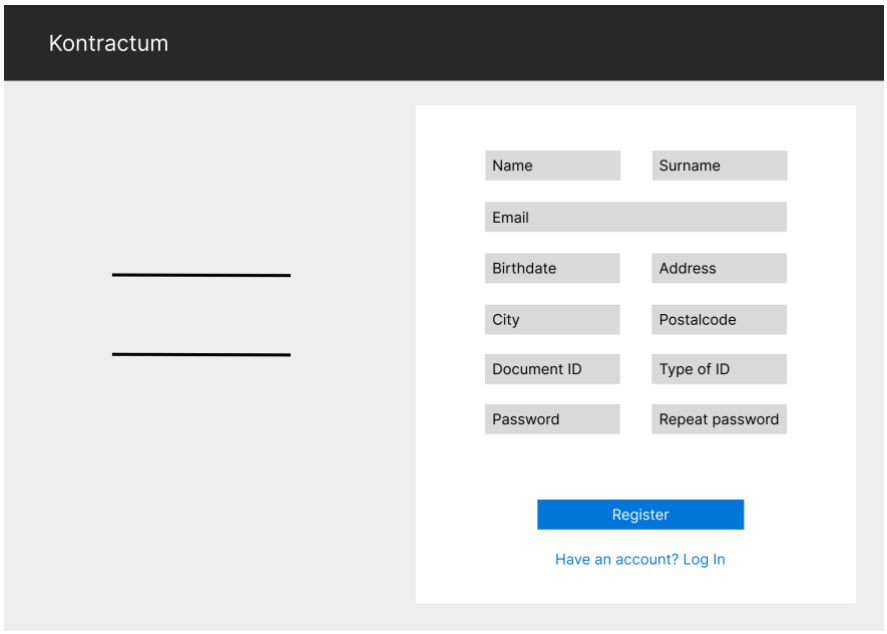

Figura 34 - Wireframe de la vista Registro

3.4 Implementación y evaluación

3.4.1 Preparación del entorno

Figura 35 - Icono de Visual Studio Code

El editor de texto seleccionado para el desarrollo del proyecto es Visual Studio Code, este potente editor de texto fué desarrollado por Microsoft, es software libre y multiplataforma. Su aceptación y uso en el mundo de la programación es muy amplio, tanto que, en una encuesta realizada a los usuarios de StackOverflow, donde más de 80000 desarrolladores participaron, Visual Studio Code fué el más votado con un 71% de uso entre los usuarios.

La estructuración de ficheros en Ruby on Rails es uno de sus puntos fuertes. El framework aboga por un uso generalizado de estructura de ficheros, esto quiere decir que todos los proyectos construidos con **RoR** contarán con una estructura de ficheros muy parecida, al contrario de otros frameworks, donde cada vez que creas un proyecto, tienes la "libertad" de estructurarlo y obtienes diferentes organizaciones en cada aplicación.

Convention over configuration.

En la siguiente tabla se muestra una explicación básica de las funciones de cada archivo y carpetas que Rails crea por defecto al iniciar una aplicación:

Archivo / Carpeta	Propósito
app/	Contiene los controladores, modelos, vistas, helpers, mailers y assets para tu aplicación. Te centrarás en esta carpeta por el resto del desarrollo.
config/	Configura las reglas de ejecución de la aplicación, rutas, base de datos y más.
config.ru	Configuración Rack para servidores basados en Rack usados para iniciar la aplicación.
db/	Contiene el esquema actual de tu base de datos, así como las migraciones de la base de datos.
doc/	Documentación detallada de tu aplicación.

Gemfile	Estos archivos te permiten especificar qué dependencias de gemas son necesitadas para tu aplicación Rails.
lib/	Módulos extendidos para tu aplicación.
log/	Archivos de Log de tu aplicación.
public/	La única carpeta vista por el mundo tal como es. Contiene archivos estáticos y assets compilados.
Rakefile	Este archivo localiza y carga tareas que pueden ser ejecutadas desde la línea de comandos. La lista de tareas son definidas a través de los componentes de Rails. En vez de cambiar el Rakefile, deberías agregar tus propias tareas, añadiendo archivos al directorio lib/tasks de tu aplicación.
script/	Contiene el script de Rails que inicia tu aplicación y contiene otros scripts usados para deployar o correr tu aplicación.
test/	Pruebas unitarias, fixtures y otras pruebas.
tmp/	Archivos temporales (como archivos de caché, pid y archivos de sesiones).
vendor/	Lugar para código de terceros. En una típica aplicación Rails, ésta incluye librerías y plugins.

Tabla 19 - Árbol de ficheros importantes

Ahora, con algunas capturas, se mostrará el árbol de directorios, así como ciertas carpetas o ficheros de interés, que son importantes o muy utilizadas durante el desarrollo del proyecto.

Estructura general:

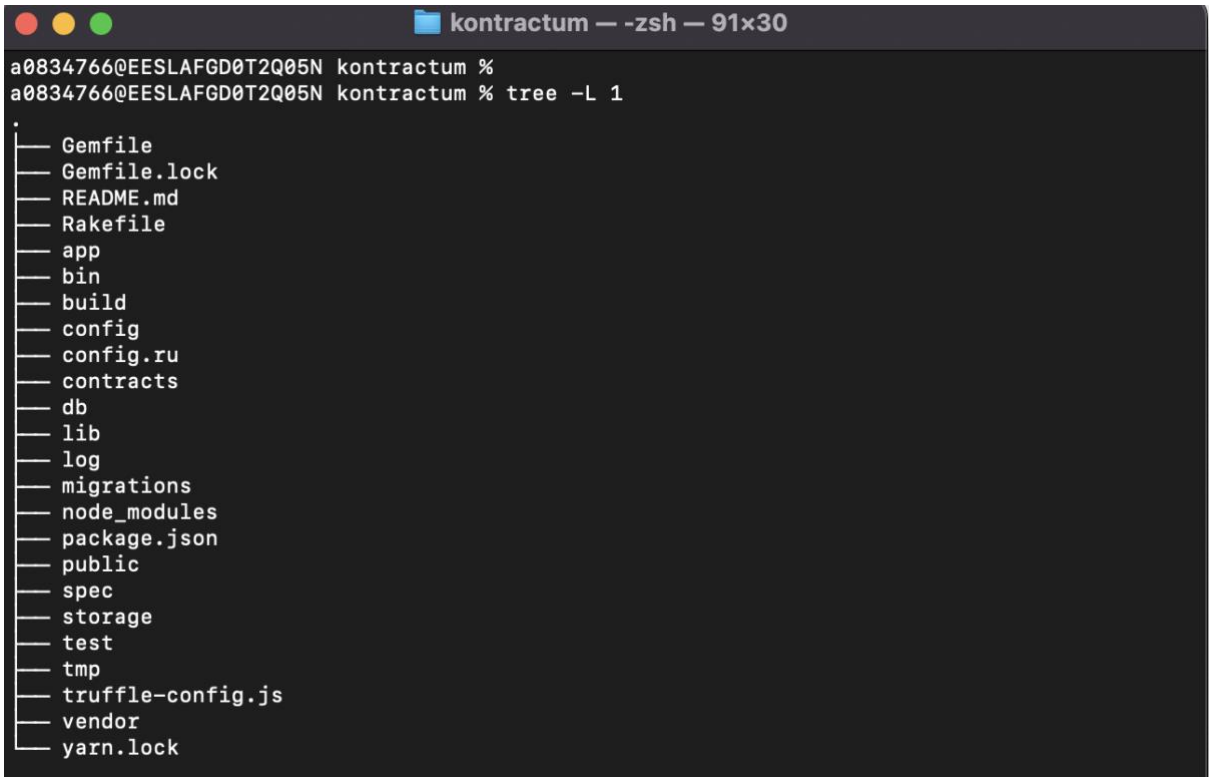

Figura 36 - Árbol de ficheros del proyecto

Dentro de la estructura del proyecto, podríamos destacar carpetas como app/, config/, db/, log/.

Dentro de app/ es donde trabajaremos el 90% del proyecto, ya que contiene directorios para el tratamiento de modelos, vistas y controladores, entre otros.

Carpeta app/ :

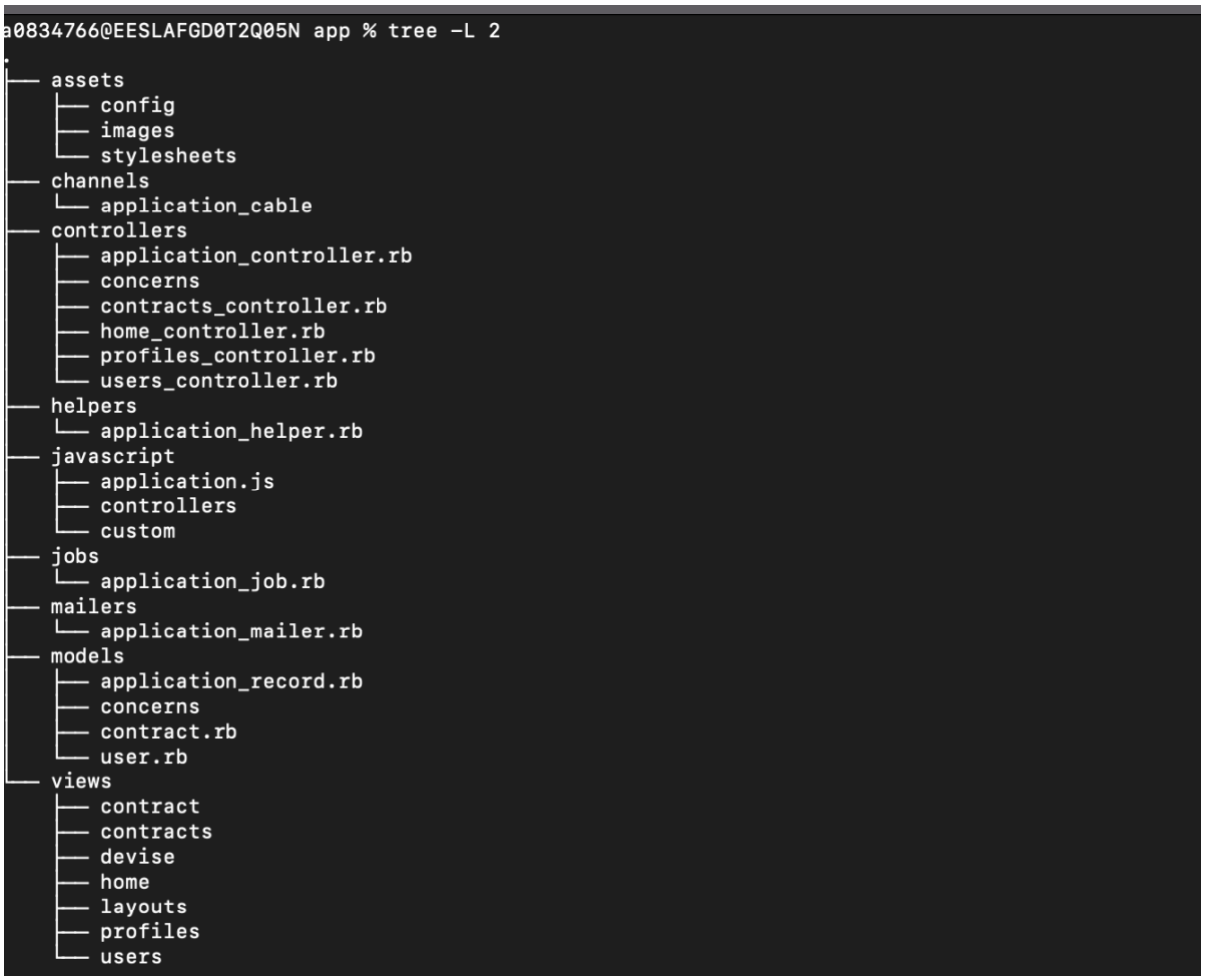

Figura 37 - Árbol de ficheros carpeta APP

En la carpeta config/, encontraremos ficheros relevantes como routes.rb, cuando hay una solicitud HTTP del usuario a la aplicación, debe dirigirse al controlador correcto. Podemos imaginar un enrutador como recepcionista que lo conecta con la persona adecuada para hablar, el fichero routes.rb se encarga de ello.

Además podemos encontrar database.yml, donde podremos indicar qué gestor de bases de datos vamos a utilizar en nuestra aplicación, por defecto viene sqlite3.

Carpeta /config:

```
a0834766@EESLAFGD0T2Q05N config % tree -L 1
.
├── application.rb
├── boot.rb
├── cable.yml
├── credentials.yml.enc
├── database.yml
├── environment.rb
├── environments
├── importmap.rb
├── initializers
├── locales
├── master.key
├── puma.rb
├── routes.rb
└── storage.yml
```

Figura 38 - Árbol de ficheros carpeta CONFIG

Dentro del directorio log/ quiero destacar un fichero llamado development.log, este fichero guarda constantemente logs de cualquier acción o request. Es de mucha utilidad para debuggear.

Carpeta log/ :

```
a0834766@EESLAFGD0T2Q05N log % tree -L 1
.
├── development.log
└── test.log
```

Figura 39 - Árbol de ficheros carpeta LOG

Finalmente, dentro de la carpeta db/ podremos encontrar archivos considerables, como las migraciones realizadas para modificar la base de datos dentro de migrate/, en schema.rb veremos un resumen del estado actual de la base de datos, este fichero no es utilizado por ActiveRecord pero, nos da una visión ordenada de como lucen las tablas y valores de nuestra base de datos.

Además nos encontramos con el fichero seeds.rb, que es de gran ayuda para generar, en nuestro caso, múltiples usuarios y contratos para poblar la base de datos y poder realizar pruebas de desarrollo.

Carpeta db/ :

```
a0834766@EESLAFGD0T2Q05N db % tree -L 1
.
├── development.sqlite3
├── migrate
├── schema.rb
├── seeds.rb
└── test.sqlite3
```

Figura 40 - Árbol de ficheros carpeta DB

Ahora centrémonos en la carpeta app/.

En la carpeta controllers encontraremos los controladores de las diferentes vistas, como users, home, contracts, y el controlador principal de la aplicación que es application_controller.rb.

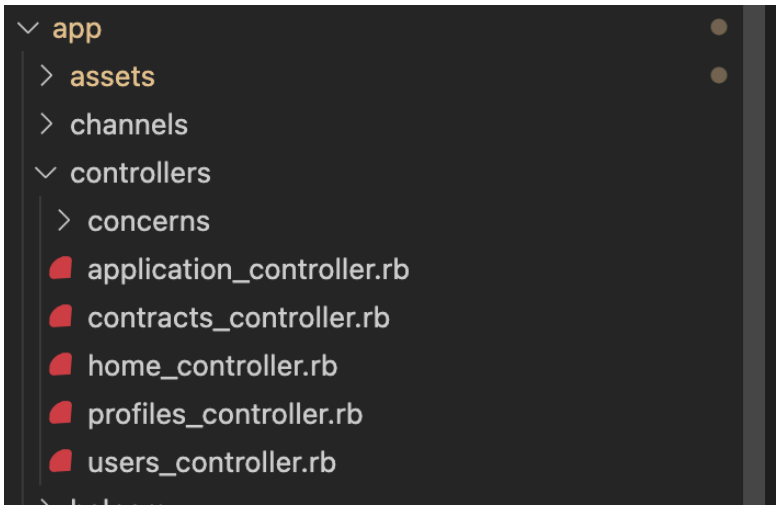

Figura 41 - Controladores

Dentro de la carpeta models encontraremos los modelos de nuestra aplicación, en este caso, los 2 modelos principales que tenemos son user.rb representando el modelo estándar de los usuarios, con sus validaciones y funciones, y el de los contratos, representado por contract.rb .

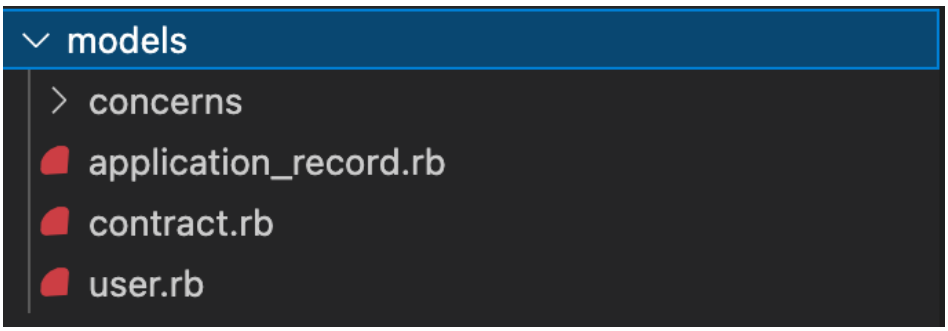

Figura 42 - Modelos

Finalmente, nos encontramos con la carpeta views. Aquí mantendremos en diferentes carpetas, las vistas que se mostrarán al usuario por pantalla, separadas en subcarpetas y mostrándose al usuario dependiendo de sus acciones en la aplicación.

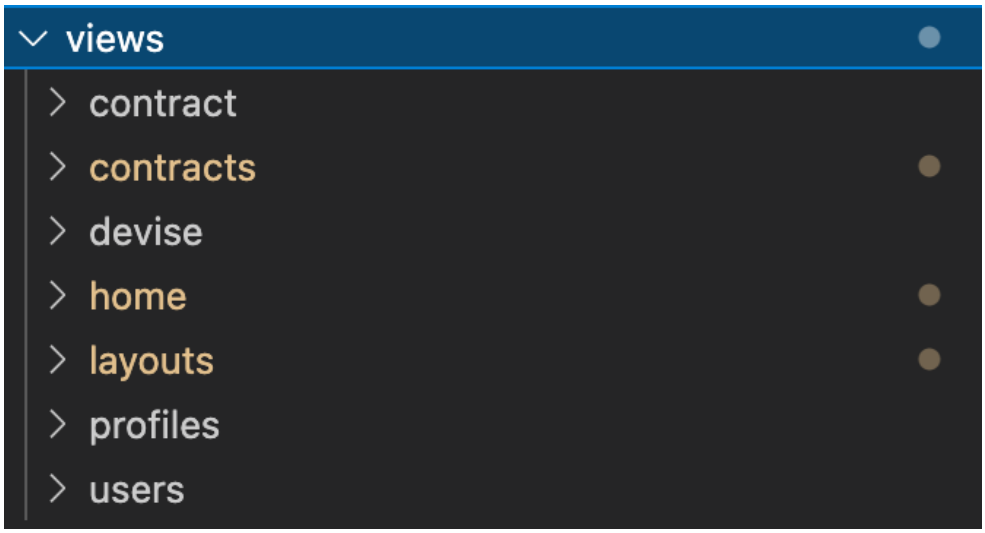

Figura 43 - Vistas

Una mención especial sobre las gemas de ruby, registradas en el fichero Gemfile, es la encargada de almacenar todas las librerías y dependencias utilizadas por el proyecto.

Las gemas de Ruby son paquetes de librerías para Ruby que se instalan en el sistema y quedan listas para ser usadas, con un simple require o con mecanismos que aporta el propio sistema de gemas para Ruby.

En la web https://rubygems.org/?locale=es podemos encontrar miles de gemas publicadas por y para desarrolladores. El fichero Gemfile de nuestro proyecto luce de la siguiente forma:

Gemfile

You, 4 weeks ago | 1 author (You)

```
1   source "https://rubygems.org"
2   git_source(:github) { |repo| "https://github.com/#{repo}.git" }
3
4   ruby "3.0.0"
5   gem "rails", "~> 7.0.3", ">= 7.0.3.1"
6   gem "sprockets-rails"
7   gem "sqlite3", "~> 1.4"
8   gem "puma", "~> 5.0"
9   gem "importmap-rails"
10  gem "turbo-rails"
11  gem "stimulus-rails"
12  gem "jbuilder"
13  gem "devise"
14  gem 'jquery-rails'
15  gem "faker"
16  gem "tzinfo-data", platforms: %i[ mingw mswin x64_mingw jruby ]
17  gem "bootsnap", require: false
18  gem "ransack"
19  gem 'kaminari'
20  gem 'bootstrap4-kaminari-views'
21  gem 'carrierwave', '~> 2.0'
22  gem "mini_magick"      You, 4 weeks ago • change pagination gem …
23
```

Figura 44 - Gemfile

Algunas de las gemas más relevantes que hemos instalado para utilizar en el proyecto son:

- Devise:

La gema devise nos facilita la implementación de la gestión del login, registro y recuperación de contraseña en nuestra aplicación. Asi como la actualización de datos de nuestros usuarios.

- Faker:

Faker nos permite acceder a una librería de ejemplos falsos de nombres, películas, canciones, direcciones y un largo etcétera, para poder generar a través de nuestro fichero seeds.rb datos falsos para poblar nuestra base de datos.

- RSpec:

Esta gema es ampliamente utilizada en el entorno de rails. Permite añadir una serie de especificaciones para que cada test describa un escenario diferente para verificar su funcionamiento.

Es una gema muy extendida que se usa de manera profesional y que permite escribir tests de una forma sencilla de leer y entender.

- Factory Bot:

Esta gema nos ayuda, junto a RSpec, a maquetar "factorías" o dummies de, por ejemplo, usuarios, para no tener que escribir o crear cada vez usuarios manualmente para realizar las pruebas, para así verificar funcionamientos de los controladores, validaciones de los modelos, etcétera.

- Ransack:

La gema Ransack es muy potente por su capacidad de búsqueda avanzada en una aplicación de Ruby on Rails. Puede crear formularios de búsqueda simples y avanzados para filtrar

- Kaminari:

No es una buena práctica mostrar la mayor parte de los datos en una página. Al trabajar con una gran cantidad de datos debemos mostrarla de forma ordenada y fácilmente accesible. La gema Kaminari nos ayuda a realizar una paginación limpia y sencilla.

- Carrierwave y mini_magick:

Carrierwave es una gema que nos permite procesar y manejar la subida de archivos a nuestra aplicación, y complementada con mini_magick, podemos validar que los archivos que subimos sean del formato que deseemos.

3.4.2 Programación e inicialización

Una vez configurada la estructura de la aplicación hace falta darle la forma al proyecto. Podemos distinguir, el diseño de los archivos html y sus correspondientes controladores (parte de cliente) y los controladores internos y la conexión a la base de datos (parte servidor).

Ruby on Rails trabaja bajo la filosofía de convención sobre configuración, por lo cual si tenemos un modelo Post, debemos tener una tabla dentro nuestro base de datos llamado posts, además en Rails se premia la agilidad del trabajo, por lo que nos ofrece líneas de comandos generadores de código para poder agilizar procesos de creación de código.

Nos referimos al scaffold, es un generador de código el cual nos permite tener las funcionalidades básicas de administración de un modelo, es decir, el CRUD (Create, Read, Update, Delete), y que son típicas para cualquier sistema transaccional.

El scaffold (en español, andamio) también nos creará las rutas correspondientes.

Una vez generados los scaffolds de los modelos necesarios, los modificaremos para crear las validaciones que creamos necesarias.

Por ejemplo:

- Si el usuario, al momento del registro, no incluye un nombre, saltará un aviso de que el formulario es inválido.
- Si el usuario introduce una fecha de nacimiento, donde su edad no se comprenda entre los 18 y 100 años, también se dará como inválido
- Si el email introducido no coincide con las expresiones regulares de un email (que contenga una arroba, y finalice en .com, .es etc…) también se dará por inválido.

Para realizar estas validaciones, Rails nos proporciona el comando validates, nos permite validar su presencia, que tenga un formato concreto, o validar una cierta condición más específica a través de una función, como el caso de la validez de edad.

Como ejemplo se mostrará una captura del archivo user.rb, contenido en app/models/

En la captura también podemos observar información sobre la anteriormente nombrada gema Devise, y ciertas relaciones con el otro modelo importante del proyecto, Contratos.

```ruby
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable

  has_many :contracts, inverse_of: 'owner'
  has_many :contracts, inverse_of: 'beneficiary'

  validates :name, presence: true
  validates :surname, presence: true
  validates :doc_id, presence: true, format: {with: /[\d]{8}[A-Z]{1}/ , message: "is invalid, please enter your id in the correct format"}
  validates :doc_kind, presence: true
  validates :email, presence: true, format: { with: URI::MailTo::EMAIL_REGEXP, message:"is invalid, please enter a valid mail"}
  validates :birthdate, presence: true, format: {with: /[\d]{1,2}(\-|\/)[\d]{1,2}(\-|\/)[\d]{4}/, message: "is invalid, please enter a valid date"}
  validates :address, presence: true
  validates :city, presence: true, format: {with: /[A-Z]\D+/, message: "is invalid, please use Uppercase in first letter"}
  validates :postalcode, presence: true, format: {with: /\b\d{5}\b/, message: "is invalid, please enter a correct Postal Code"}

  validate  :is_adult?

  enum :doc_kind, {dni: 0, driver_license: 1, passport: 2}
  enum :role, {admin: 10, customer: 20, premium_customer: 30, blocked: 1, banned: 0}, default: :customer

  def is_adult?
      my_bday = Date.parse(birthdate) rescue nil      You, 2 months ago • fix is_adult validation …
      return false if my_bday.blank?

      if !(((Date.today - 18.year) >= my_bday) && (Date.today - 100.year) <= my_bday)
        errors.add(:birthdate, "User is not adult")
      end

  end
end
```

Figura 45 - Modelo User

Una vez creados los modelos, crearemos los tests de cada uno de ellos.

Con los test, verificaremos el funcionamiento y la integridad de los modelos, poniendo nuestras validaciones a prueba mediante la creación de distintos escenarios. se mostrarán algunos de los tests ya que, validar y probar toda la información es un proceso bastante largo:

spec > models > user_spec.rb
You, last month | 1 author (You)

```ruby
require "rails_helper"

RSpec.describe User, type: :model do

  context "A user isn't valid without... " do

    it "valid attributes" do
      valid_user = build(:user)
      expect(valid_user).to be_valid
    end

    it "having a valid name" do
      non_valid_user = build(:user, name: nil)
      expect(non_valid_user).to_not be_valid
    end

    it "having a valid surname" do       You, last month • bett
      non_valid_user = build(:user, surname: nil)
      expect(non_valid_user).to_not be_valid
    end

    it "having a valid doc_id" do
      non_valid_user = build(:user, doc_id: "36R5h7")
      expect(non_valid_user).to_not be_valid
    end

    it "having a valid email" do
      non_valid_user = build(:user, email: "bad#email.what")
      expect(non_valid_user).to_not be_valid
    end
```

Figura 46 - Tests para modelo User

El generador de código scaffold, también nos genera unas funciones CRUD, básicas, pero funcionales para trabajar con nuestros modelos ya generados. Podemos crear, modificar y borrar usuarios de nuestro sistema.

En los controladores definiremos las acciones a realizar dependiendo del request:

- GET – Obtener datos sobre usuario / contrato
- POST – Crear un nuevo usuario / contrato

- DELETE – Eliminar un usuario / contrato
- UPDATE – Cambiar un elemento usuario / contrato

Por ejemplo, la función /user/POST se ve tal que así:

```ruby
# POST /users or /users.json
def create
  @user = User.new(user_params)

  respond_to do |format|
    if @user.save
      format.html { redirect_to user_url(@user), notice: "User was successfully created." }
      format.json { render :show, status: :created, location: @user }
    else
      format.html { render :new, status: :unprocessable_entity }
      format.json { render json: @user.errors, status: :unprocessable_entity }
    end
  end
end
```

Figura 47 - Función creación usuarios

Y la función /contract/POST/ se vería de la siguiente forma:

```ruby
# POST /contracts or /contracts.json
def create
  @contract = Contract.new(contract_params.except(:beneficiary))

  @contract.beneficiary = User.find(contract_params[:beneficiary].to_i)

  @contract.owner = current_user

  respond_to do |format|
    if @contract.save
      format.html { redirect_to user_contracts_url(current_user), notice: "Contract was successfully created." }
      format.json { render :show, status: :created, location: @contract }
    else
      format.html { render :new, status: :unprocessable_entity }
      format.json { render json: @contract.errors, status: :unprocessable_entity }
    end
  end
end
```

Figura 48 - Función creación de contratos

Una vez implementadas las funciones CRUD, y siguiendo con la metodología TDD, se crean los tests necesarios para comprobar que todas las funciones de creación, modificación y borrado son funcionales.

Los tests de los controladores tienen la siguiente estructura:

- Con la ayuda de FactoryBot, creamos un usuario “dummy”, con atributos válidos

```ruby
FactoryBot.define do
  factory :user do
    name {"Paco"}
    surname {"Gonzalez"}
    doc_id {"35354647N"}
    doc_kind {0}
    sequence(:email) {|n| "email#{n}@gmail.com"}
    birthdate {"3/5/1999"}
    address {"Calle San Onofre"}
    city {"Torrent, Valencia"}
    postalcode {"46900"}
    password {"xxxxxx"}
    role {20}
  end

  factory :admin, class: "user"  do
    name {"Jorge"}
    surname {"Dominguez"}
    doc_id {"78564539N"}
    doc_kind {0}
    sequence(:email) {|n| "email#{n}@gmail.es"}      You, last month • advanc
    birthdate {"1/5/1999"}
    address {"Calle Don Patricio"}
    city {"Albaida, Valencia"}
    postalcode {"46700"}
    password {"xxxxxx"}
    role {10}
  end
end
```

Figura 49 - Factorías de usuarios

- En el fichero de pruebas de spec, creamos una instancia del usuario dummy y creamos también dos objetos, :valid_attributes e :invalid_attributes para crear escenas donde el usuario dummy cree contratos válidos y no válidos, esperando una buena respuesta por parte de nuestras funciones creadas anteriormente.

```ruby
require 'rails_helper'

RSpec.describe "/contracts", type: :request do
  let(:new_user){ create(:user) }

  let(:beneficiary){ create(:user) }

  let(:valid_attributes){ attributes_for(:contract).merge(beneficiary: beneficiary.id) }

  let(:invalid_attributes) do {
    title: "",
    desc: "",
    kind: "",
    status: :drafted,
    owner: nil,
    beneficiary: beneficiary.id,
    payable: true
  } end
```

Figura 50 - Tests Requests Contratos

- Las pruebas creadas en los tests, comprueban si, por ejemplo, un usuario genera un contrato, este reciba una respuesta por parte de la aplicación indicando que la operación se ha realizado correctamente.

```ruby
describe "POST /create" do
  context "with valid parameters" do
    it "creates a new Contract" do
      sign_in new_user
      expect {
        post user_contracts_path(new_user), params: { contract: valid_attributes }
      }.to change(Contract, :count).by(1)
    end

    it "redirects to the created contract" do
      sign_in new_user
      post user_contracts_url(new_user), params: { contract: valid_attributes }
      expect(response).to redirect_to(user_contracts_url(new_user))
    end
  end
```

Figura 51 - Tests Requests Contratos

- Otros casos serían, por ejemplo, comprobar que la aplicación renderiza una respuesta exitosa tras el usuario, realizar ciertas acciones, como ver la lista de contratos de nuestro usuario, recibir una respuesta exitosa tras crear un contrato correctamente, o entrar a la página correcta si queremos editar cierto contrato.

```ruby
describe "GET /index" do
  it "renders a successful response" do
    sign_in new_user
    valid_contract = create(:contract)
    get user_contracts_path(valid_contract.owner)
    expect(response).to be_successful
  end
end

describe "GET /show" do
  it "renders a successful response" do
    sign_in new_user
    valid_contract = create(:contract)
    get user_contracts_url(valid_contract)
    expect(response).to be_successful
  end
end

describe "GET /new" do
  it "renders a successful response" do
    sign_in new_user
    valid_contract = create(:contract)
    get new_user_contract_path(valid_contract.owner)
    expect(response).to be_successful
  end
end

describe "GET /edit" do
  it "renders a successful response" do
    sign_in new_user
    valid_contract = create(:contract)
    get edit_user_contract_path(valid_contract.owner, valid_contract.id)
    expect(response).to be_successful
  end
end
```

Figura 52 - Tests Requests Contratos

Entre otras, estas son algunas de las pruebas que han sido escritas para verificar la integridad de las funciones sobre usuarios y contratos, estando las funciones CRUD cubiertas en un 100% por sus correspondientes tests.

Una vez programadas las funciones básicas cubiertas y operativas, pasamos a mejorar la UI del sistema, la cual será cubierta en la parte 4 durante el tour de la aplicación.

3.4.3 Tecnologías blockchain

Para realizar la integración de la blockchain con nuestro sistema en desarrollo, se ha escogido Truffle Suite.

Truffle Suite se trata de un entorno de desarrollo, un conjunto de herramientas que permiten a los desarrolladores realizar un testing e integración continua de sus dApps de una forma eficiente y sencilla, gestionando el ciclo de vida de los smart contracts desde su diseño hasta su despliegue en la blockchain.

Truffle Suite aporta las siguientes herramientas: Truffle, Ganache, Drizzle, Truffle Boxes y Truffle Teams, pero en nuestra aplicación tan solo utilizaremos 2 de ellas, Truffle y Ganache.

Truffle es el framework más popular de Ethereum, con la misión de hacerle la vida más fácil a los desarrolladores en el proceso de integración y testing de dApps (decentralized Applications). Para ello, aporta las siguientes funcionalidades:

- Compilación, enlazado, despliegue y administración binaria de smart contracts.
- Testeo automático de smart contracts para un rápido desarrollo de los mismos.
- Aporta un entorno de implementación y migración extensible y programable.
- Trabajar con diferentes redes, tanto públicas como privadas.
- Consola interactiva para realizar una comunicación directa con los smart contracts.
- Permite ejecutar scripts externos que interactúen con los smart contracts.

Ganache es una cadena de bloques personal para el desarrollo rápido de dApps de Ethereum. Permite desarrollar, implementar y probar las dApps en un entorno seguro y determinista.

Ganache UI es una aplicación de escritorio que admite la tecnología Ethereum.

Esta red de pruebas local, nos permite desplegar en la blockchain nuestros smart contracts creados sin ningún tipo de coste, ya que la aplicación nos otorga éter (ETH) de prueba, moneda de curso utilizada en la blockchain de Ethereum, para pagar comisiones a los nodos validadores por interactuar con la blockchain.

También se utilizará la extensión de navegador o plugin, Metamask.

MetaMask es una extensión o plugin para navegadores web que permite a los usuarios interactuar fácilmente con las dApps de la blockchain de Ethereum. Esto es posible, porque MetaMask hace de puente entre las dApps y los navegadores web facilitando el uso y disfrute de las mismas.

Para ello, MetaMask usa la interfaz y API web de Ethereum, **web3.js**. Esta librería oficial de Ethereum sería la base fundamental del mundo de posibilidades ofrecidas por MetaMask. Gracias a ella sería posible crear un proxy o puente comunicacional entre las dApps, MetaMask y los usuarios.

Ahora, se explicará la integración entre blockchain y la aplicación.

3.4.4 Integración de blockchain en el proyecto

Primero de todo, se deberá iniciar Truffle en nuestro proyecto, para ello debemos instalar truffle e iniciarlo.

Se generará un archivo llamado **truffle-config.js**, se trata de un archivo de configuración y se encuentra en la raíz del directorio del proyecto. Este archivo es un archivo Javascript y puede ejecutar cualquier código necesario para crear la configuración. El objeto que se exportará representa la configuración del proyecto y tendrá el siguiente aspecto.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*",
    },
  },
  compilers: {
    solc: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
      version: "^0.5.0",
    },
  },
};
```

Figura 53 - Aspecto truffle-config.js

Tras poner en marcha el proyecto, descargamos e iniciaremos Ganache, nuestra blockchain local.

Para enlazar Ganache con nuestro proyecto truffle, basta con crear un nuevo workspace, ir al apartado de contratos y dar click al botón de enlazar proyecto truffle.

Añadimos el proyecto, seleccionando el archivo **truffle-config.js**, creado cuando iniciamos Truffle en nuestro proyecto, y ya tendremos la conexión lista.

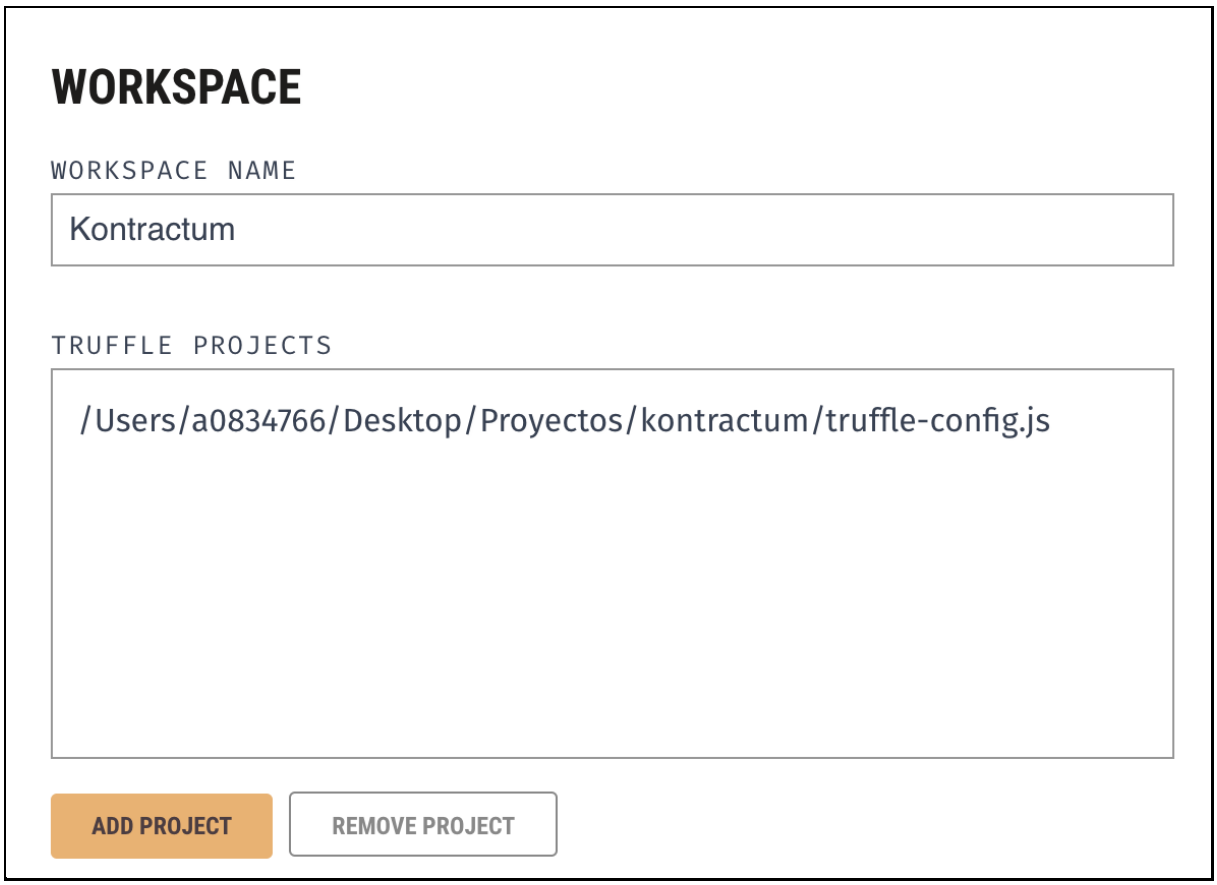

Figura 54 - Enlazar Ganache con Proyecto

Una vez realizada la conexión entre blockchain y proyecto, realizaremos la programación del Smart Contract para desplegarlo en la blockchain.

El Smart Contract se escribe en el lenguaje Solidity. Es un lenguaje de alto nivel orientado a contratos. Su sintaxis es similar a la de JavaScript y está enfocado específicamente a la Máquina Virtual de Ethereum (EVM, Ethereum Virtual Machine).
Se trata de un lenguaje tipado de manera estática y acepta, entre otras cosas, herencias, librerías y tipos complejos definidos por el usuario.

El contrato inteligente inicial tiene una estructura muy sencilla. Tendremos una estructura básica de contrato, el cual contiene un ID y un título, y una función de creación de contrato, el cual asignará un valor al ID del contrato y el título otorgado por el usuario. Los contratos al ser creados se guardan en un array.

Una vez programado el Smart Contract, compilaremos y migraremos nuestro contrato a la blockchain. Si hemos cometido algún error o tenemos algún bug en el código del Smart Contract, en el momento de compilación nos avisará de los errores.

Con el comando **truffle compile**, se compilarán todos los contratos que hayamos programado.

Y con el comando **truffle migrate**, si la compilación ha sido exitosa, se migrarán a la blockchain y podremos empezar a interactuar con el contrato inteligente ya desplegado.

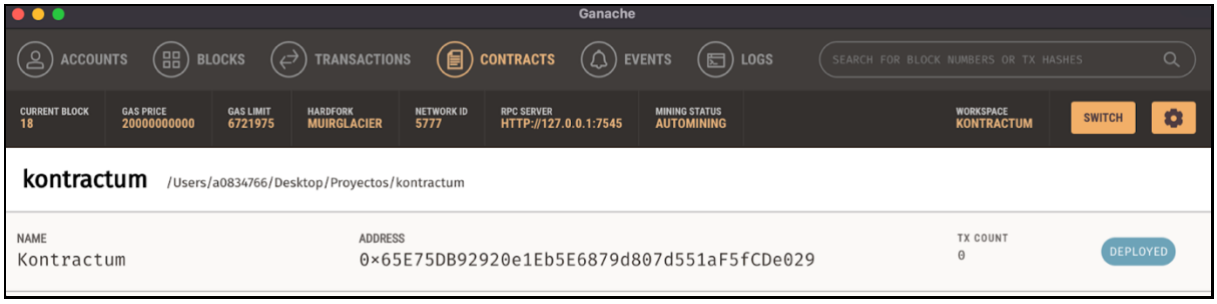

Figura 55 - Smart Contract desplegado en Ganache

Podemos verificar que el proceso se realizó correctamente en nuestro cliente Ganache, donde, si nos dirigimos al apartado de "Contracts", podremos ver nuestro contrato con su dirección asociada. Si hacemos click en el contrato, podremos ver información relacionada con este, como la cantidad de datos que tiene almacenados, cuantas transacciones se han realizado al contrato, o información sobre el evento de su última migración.

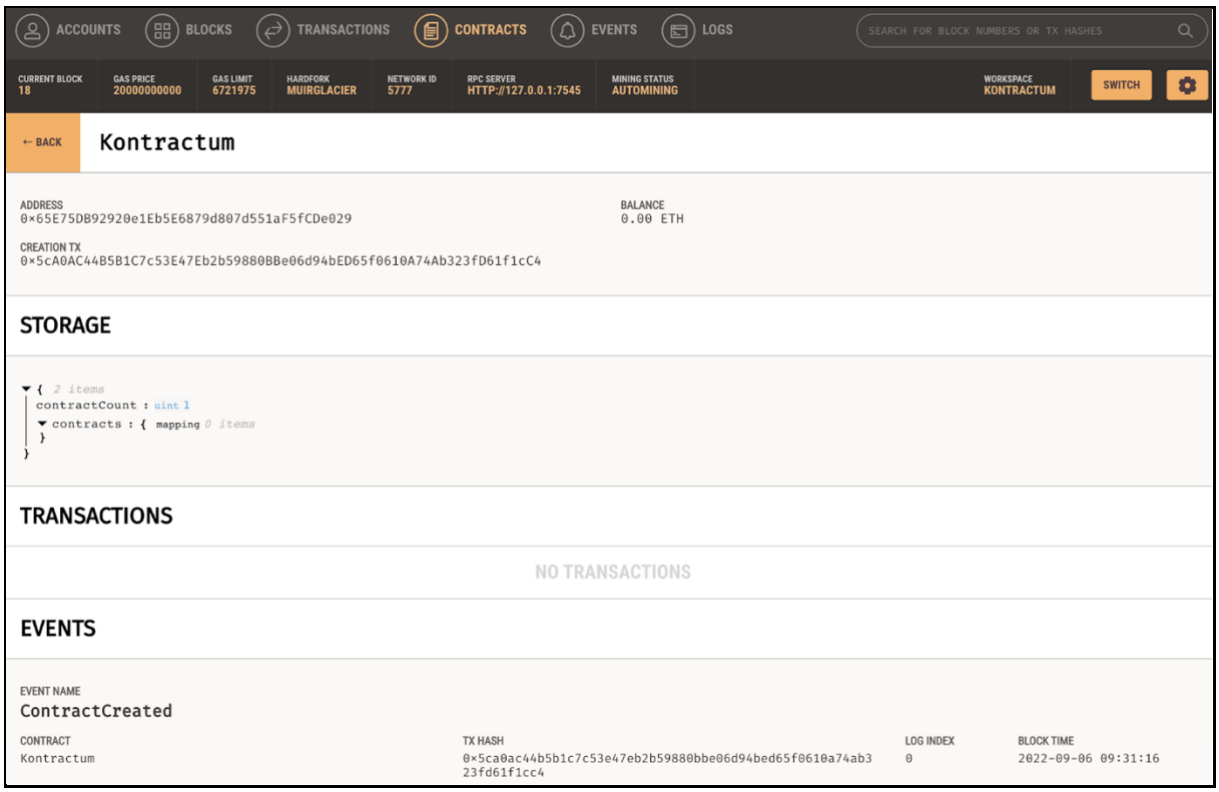

Figura 56 - Información del Smart Contract desplegado

Listo, ahora nuestra blockchain ya tiene cargada el Contrato Inteligente que hemos creado, el siguiente paso será que, a través de MetaMask, cuando un usuario realice cierta acción sobre los contratos en la aplicación web, ya sea creación, destrucción o modificación, se genere una transacción a la blockchain guardando en ella datos e información necesaria sobre el contrato.

El primer paso será descargar e instalar el plugin web de Metamask en nuestro navegador. Una vez instalada y creada nuestra cuenta, podremos ver algo similar a esto.

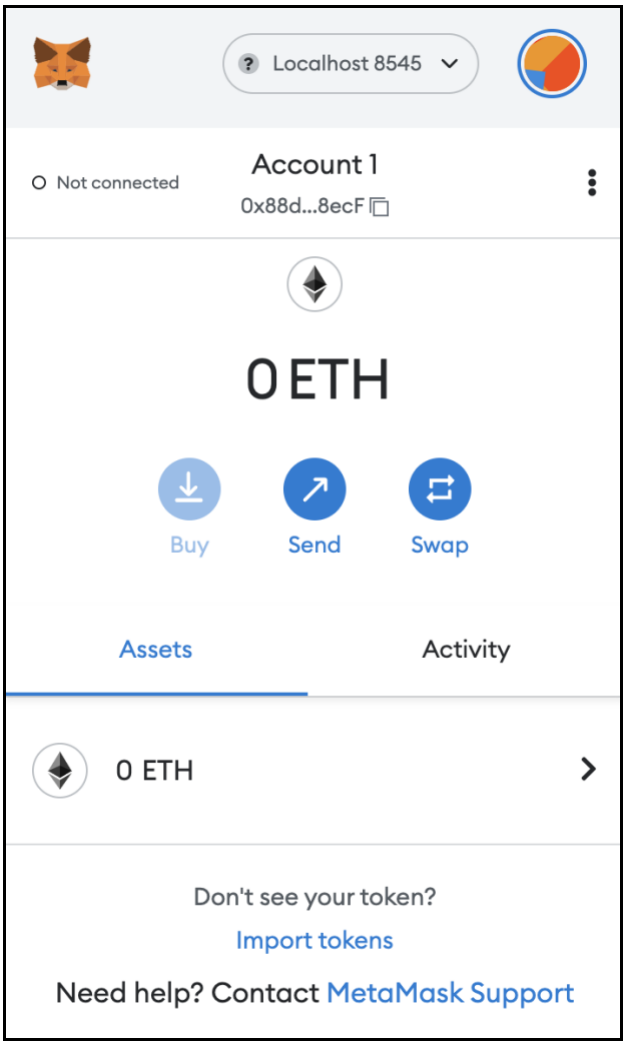

Figura 57 - Vista principal de Metamask

De arriba a abajo, nos encontramos con un desplegable donde podemos elegir a qué red conectarnos, por defecto, se encuentra conectada a la red principal de Ethereum, también podemos cambiar y conectarnos a las redes de desarrollo, o incluir una red privada nuestra, que es la opción que vamos a escoger.

También vemos información sobre nuestra cuenta, el string Ox88…8ecf simboliza nuestra clave pública, esta clave nos “representa” dentro de la blockchain.

Por último, podemos apreciar como tenemos un saldo en ETH, con tres botones para enviar, recibir o cambiar nuestro ETH.

Para poder conectar nuestro plugin a la red blockchain de Ganache, basta con rellenar los datos con la información proporcionada por Ganache sobre la red.

El dato más importante en este caso sería la red RPC. Las siglas RPC vienen del inglés, y significan una llamada de procedimiento remoto, lo cual es un conjunto de protocolos que permiten que un cliente (como MetaMask) interactúe con una cadena de bloques.

En el caso de nuestra red Blockchain sería la siguiente dada por Ganache:

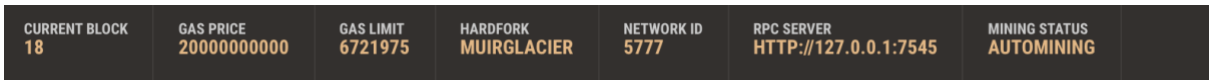

Ilustración 56. RPC Server de Ganache

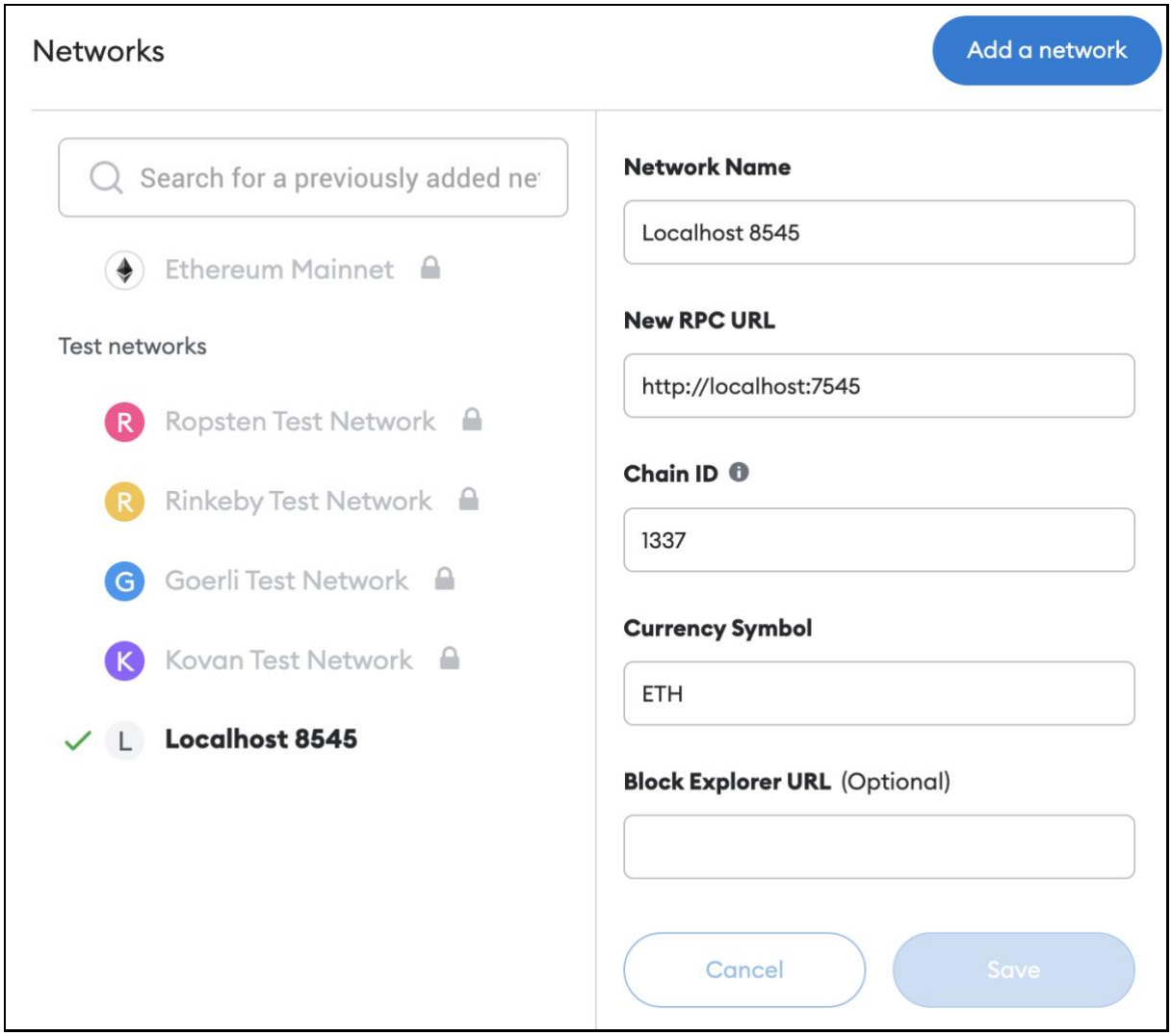

Figura 58 - Configuración de red local en Metamask

Una vez configurada correctamente nuestra red local. Metamask ya estará conectada a nuestra blockchain privada de desarrollo.

Por último, la aplicación deberá informar a Metamask sobre cuándo realizar una transacción. Para ello, el estado del contrato deberá de ser “tokenizable”, que el rol del usuario le permita interactuar con la blockchain y tener el plugin de Metamask instalado. En caso de no cumplir alguna de ellas, se ha dotado al sistema con los avisos e indicaciones necesarias para poder operar con blockchain.

Una vez el usuario haga clic en el botón, llamará a un script de Javascript cargado en la aplicación, que se encargará de:

- Recopilar la cuenta de metamask del usuario.
- Obtener el email del usuario que ha firmado.
- Obtener el hash resultante de pasar el binario del contrato por el algoritmo SHA256.
- Llamar a la función de crear contrato del Smart Contract cargado en la blockchain.
- Manejar el recibo de la transacción, haya sido correcta o incorrecta, y avisar al usuario a través de un pop-up.

Una vez la aplicación detecta que ambas firmas de las dos partes del contrato han sido transferidas, el estado del contrato pasa al estado “Tokenized”.

El flujo final sería:

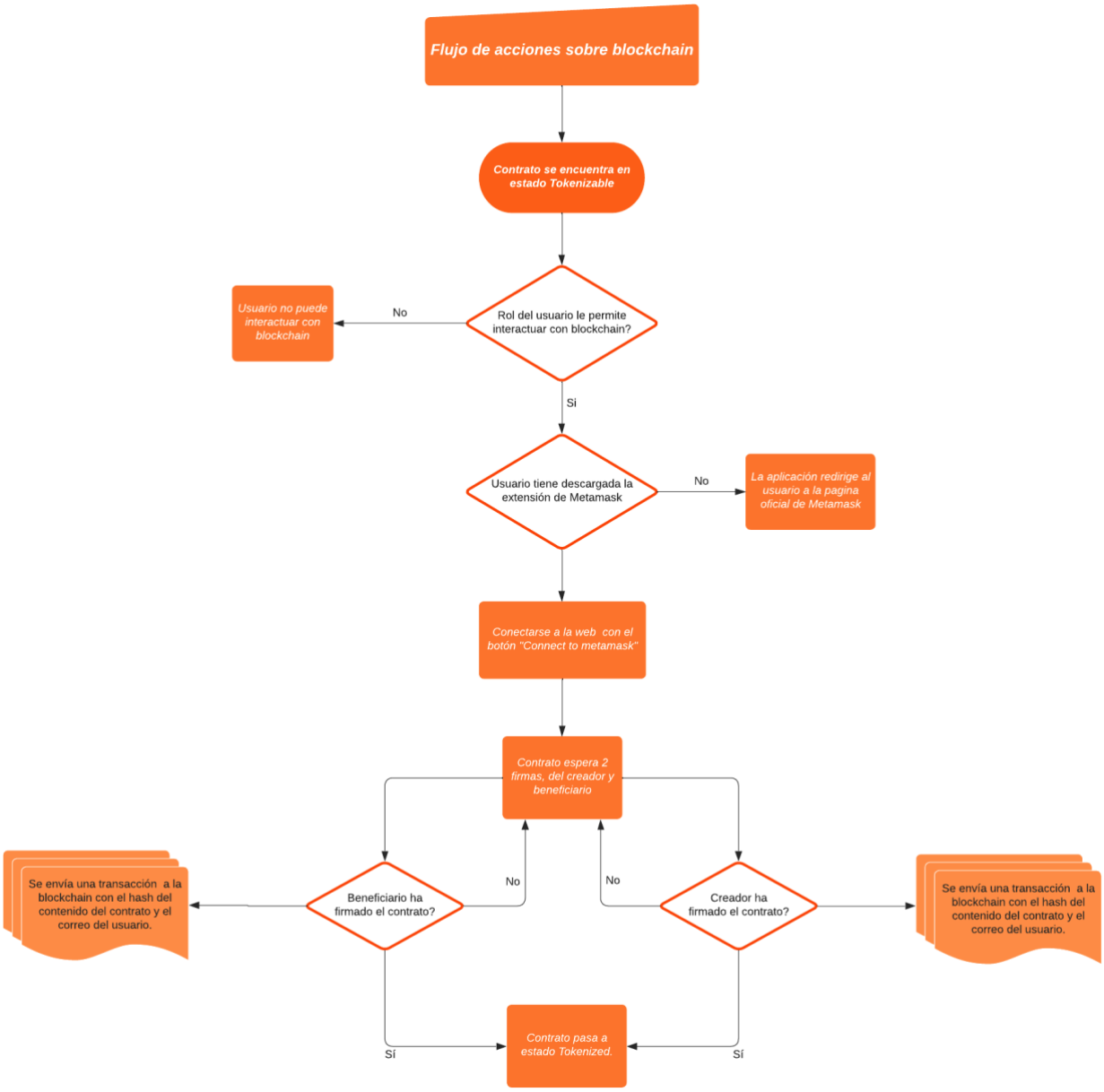

Figura 59 - Proceso operación con blockchain

4 Resultado del proyecto

4.1 Tour de la aplicación

En este apartado se van a mostrar el resultado final de la implementación mediante capturas de los diferentes apartados y funcionalidades del sistema.

El primer paso para acceder a la aplicación web es el inicio de sesión, en esta ventana se deben introducir el nombre de usuario y la contraseña, en el caso de que se introduzca mal cualquiera de los

dos campos, se mostrará una alerta del campo introducido erróneamente, y se restablecerá el campo de contraseña. En cambio, si los datos son correctos, accederá a la aplicación.

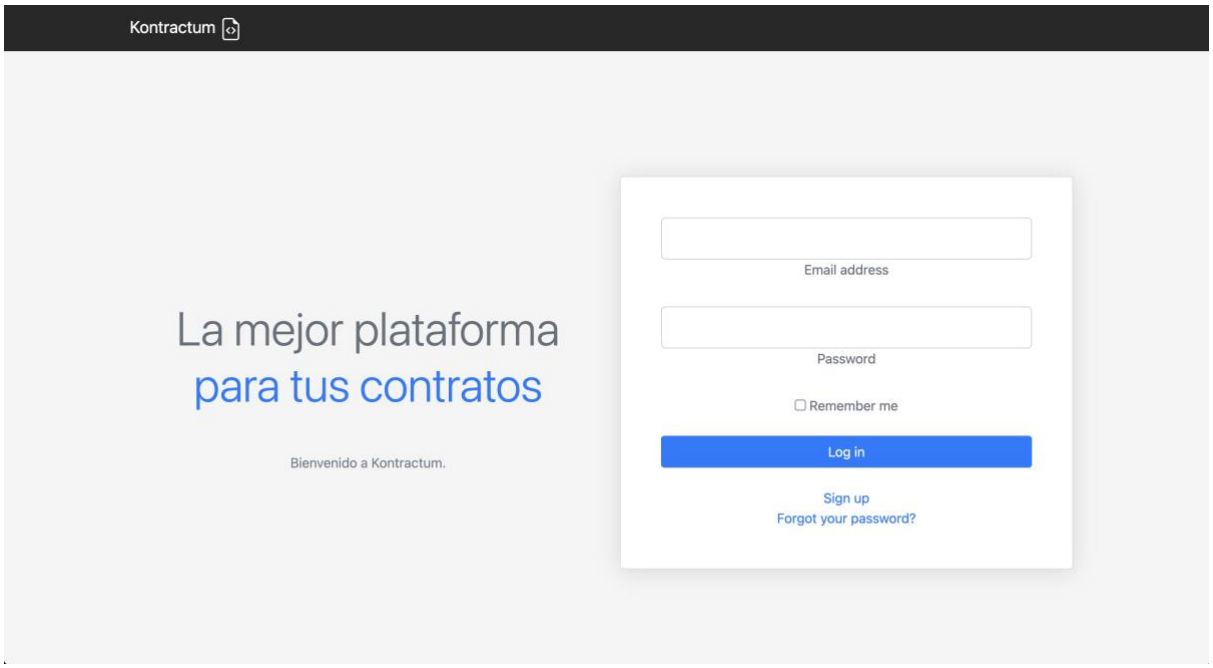

Figura 60 - Landing page oficial

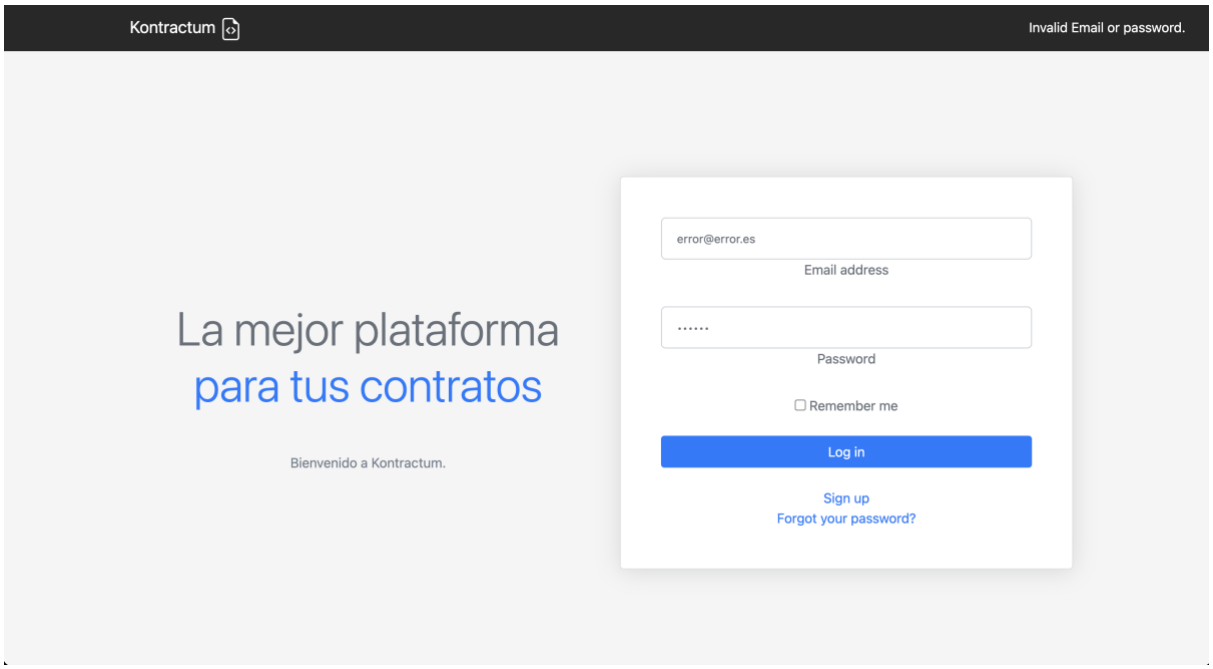

Figura 61 - Error en el login (Arriba derecha)

Si el usuario no dispone de cuenta en la plataforma, tras hacer clic en el botón e “Sign up” se le redirigirá a la vista de registro. Todos los campos en el son obligatorios, en caso de no rellenar correctamente o dejar alguno en blanco el usuario será avisado por medio de una alerta.

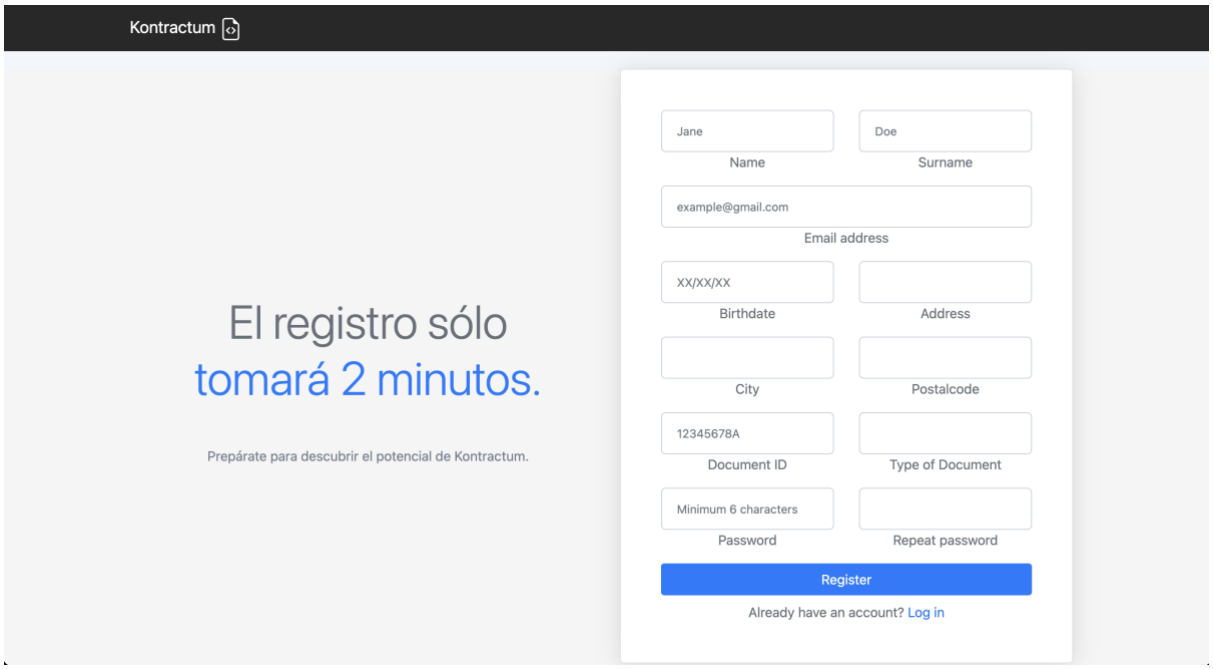

Figura 62 - Registro de usuario

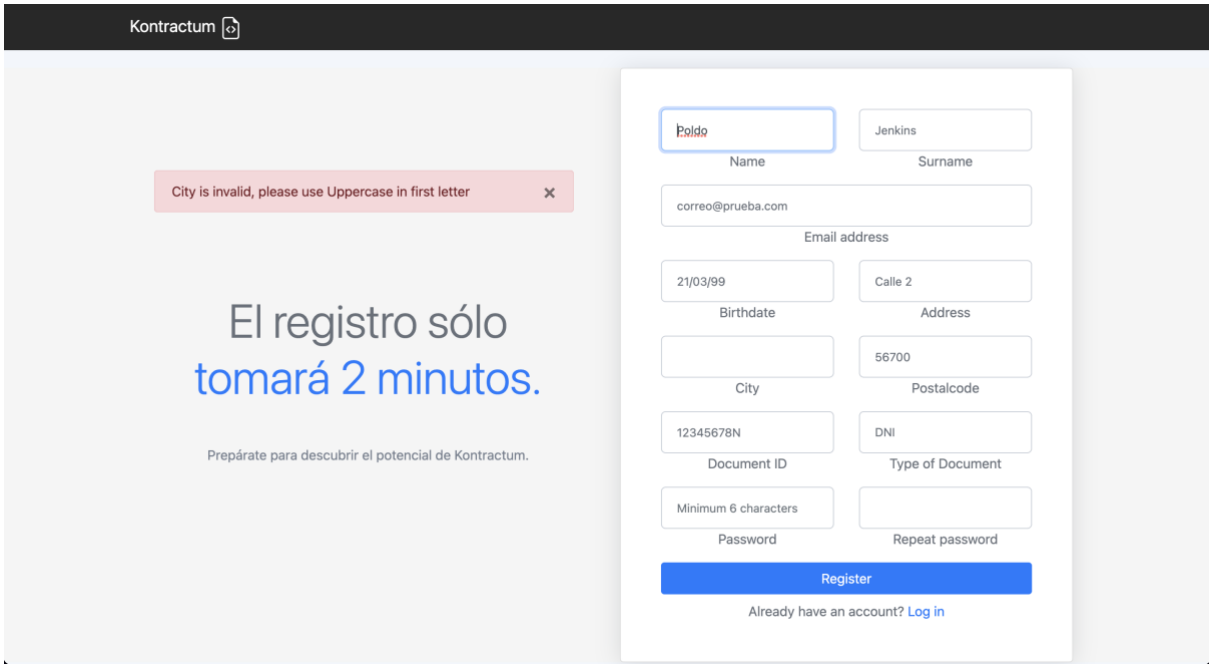

Figura 63 - Error en el registro

La plataforma también cuenta con un sistema de recuperación de contraseña, en el caso de no poder acordarse, al usuario se le enviará un correo con instrucciones para su recuperación.

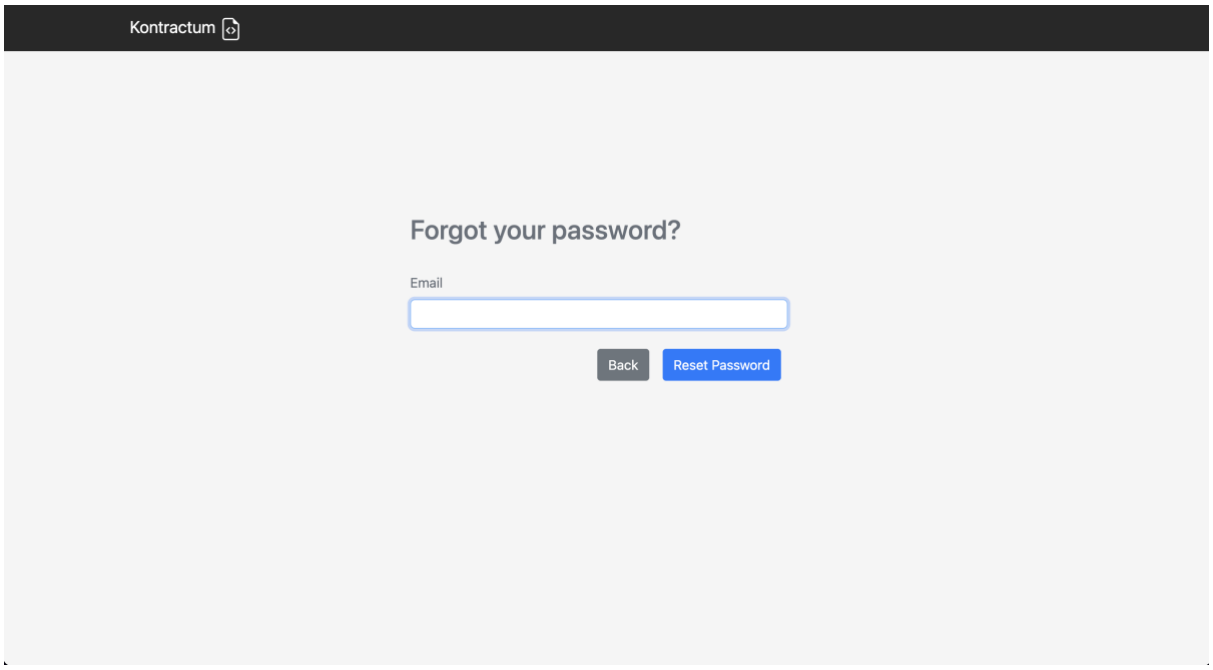

Figura 64 - Recuperar contraseña

Tras un inicio de sesión exitoso, el usuario será redirigido a la pantalla principal de la aplicación, también llamada Home.

En esta vista aparecerán todos los contratos en los cuales participemos, en caso de no participar en ninguno, como le ocurre a este nuevo usuario, se nos indica mediante un mensaje que los contratos se listarán ahí. También incorpora un formulario de búsqueda, donde podemos filtrar contratos basándonos en su título, descripción o tipo (Kind) que actúa como una especie de tag, junto con un botón de reset para limpiar el campo de búsqueda.

En la zona izquierda podemos distinguir dos zonas, la primera, el botón de Crear Contrato, y la segunda, la columna de filtros. En esta última, podemos filtrar dependiendo del estado del contrato o si somos beneficiarios u creadores del contrato.

En el header, vemos 6 acciones, Home, Dashboard, Edit Profile, Edit Password, Connect to Metamask y LOG OUT.

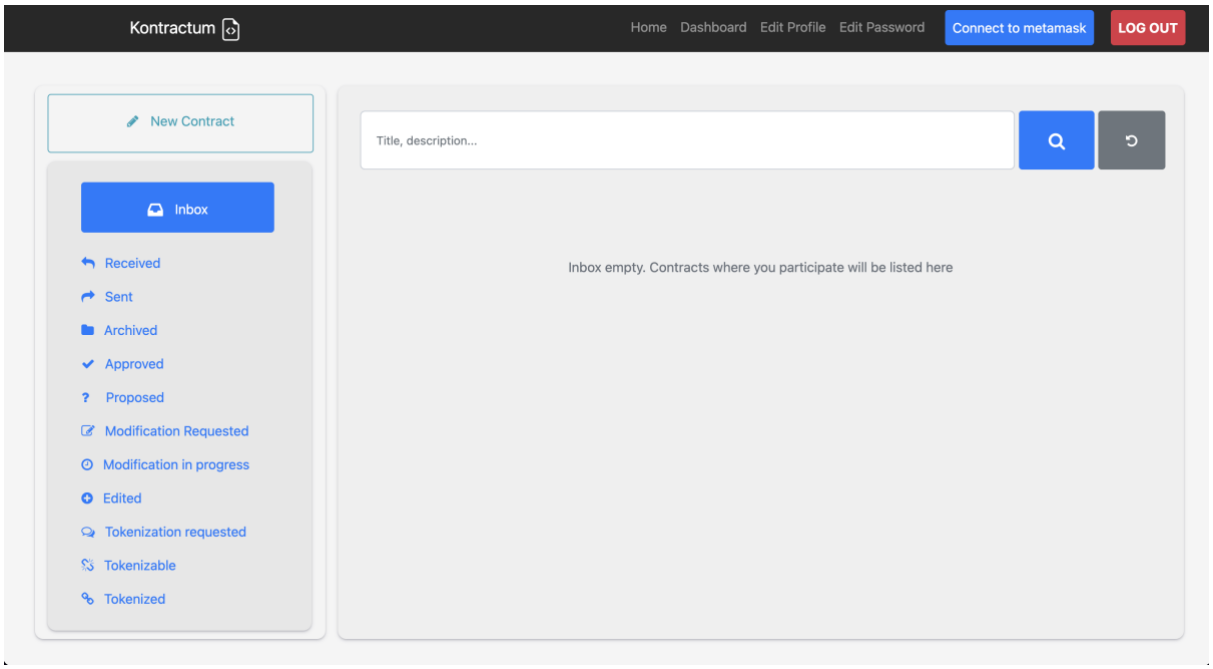

Figura 65 - Pantalla principal de la aplicación

La siguiente ilustración es la vista de crear contrato, en ella debemos indicar el título, descripción, tipo, pagabilidad y el beneficiario, así como subir nuestro documento u contrato. Los contratos, por simpleza de validaciones, solo pueden subirse en formato PDF, por motivos de seguridad.

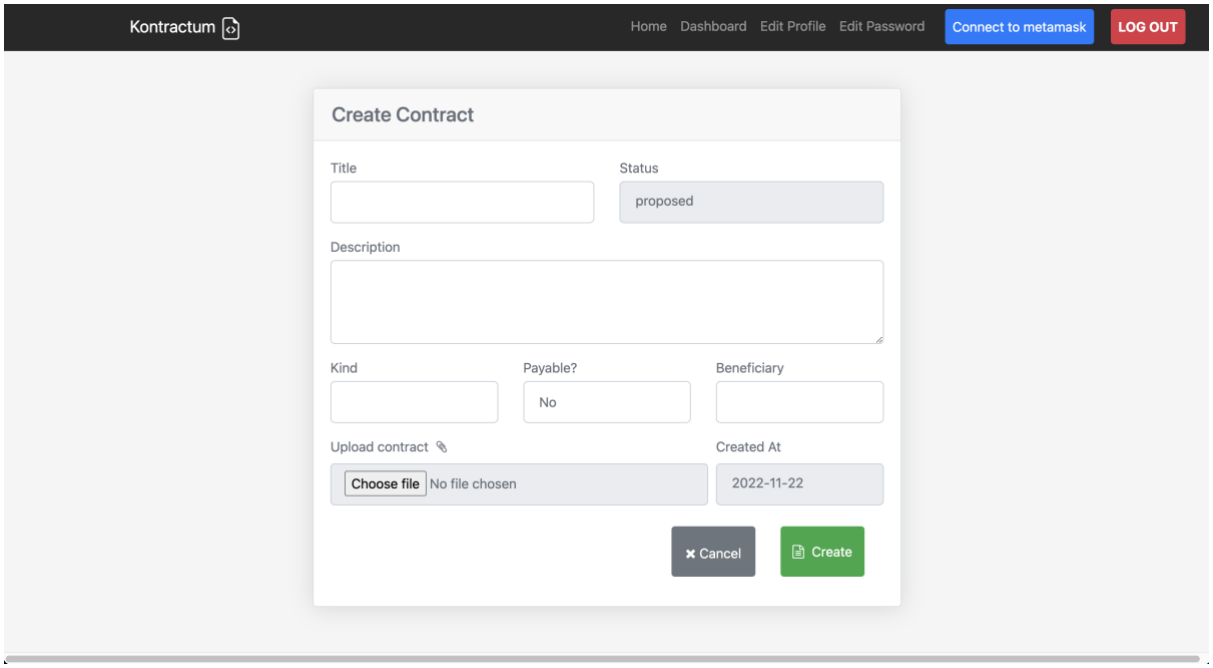

Figura 66 - Página creación de contrato

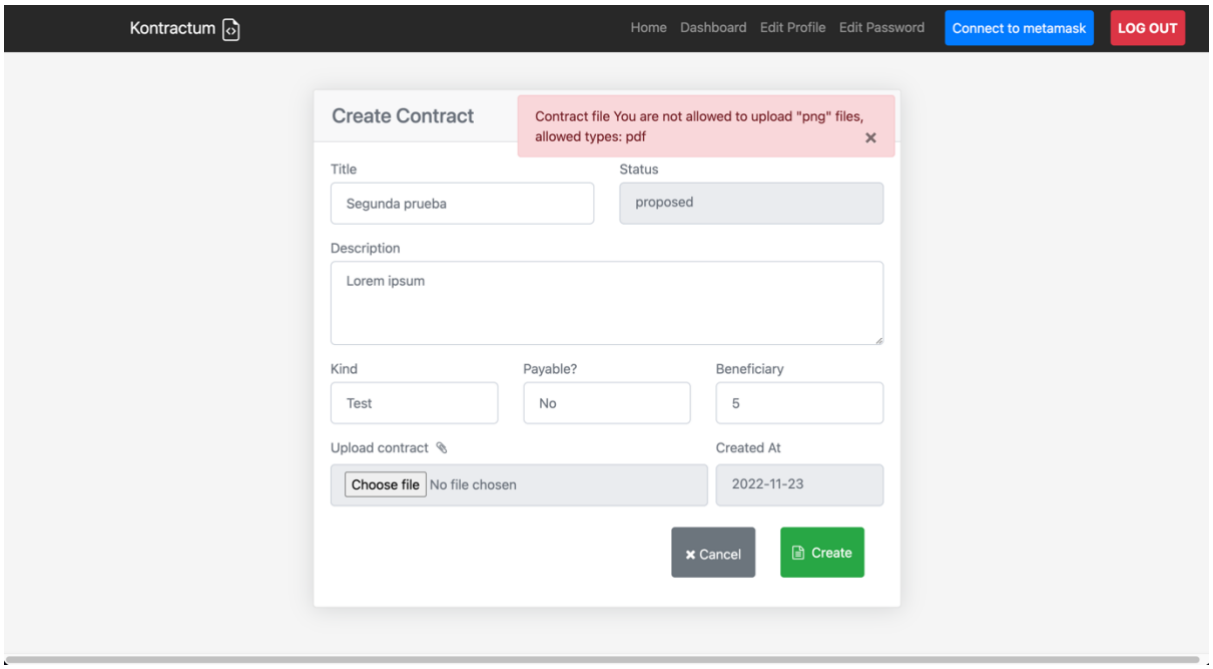

Figura 67 - Error por formato de documento

Tras crear o participar en contratos, nos aparecen listados en orden de creación (más recientes primero) en nuestra bandeja de entrada. También podemos ver tags indicando el tipo de dato de la columna, título, descripción, estado y fecha de creación. Si tenemos acciones pendientes con algún contrato, nos aparece en el mismo un botón de acción requerida (Action needed!).

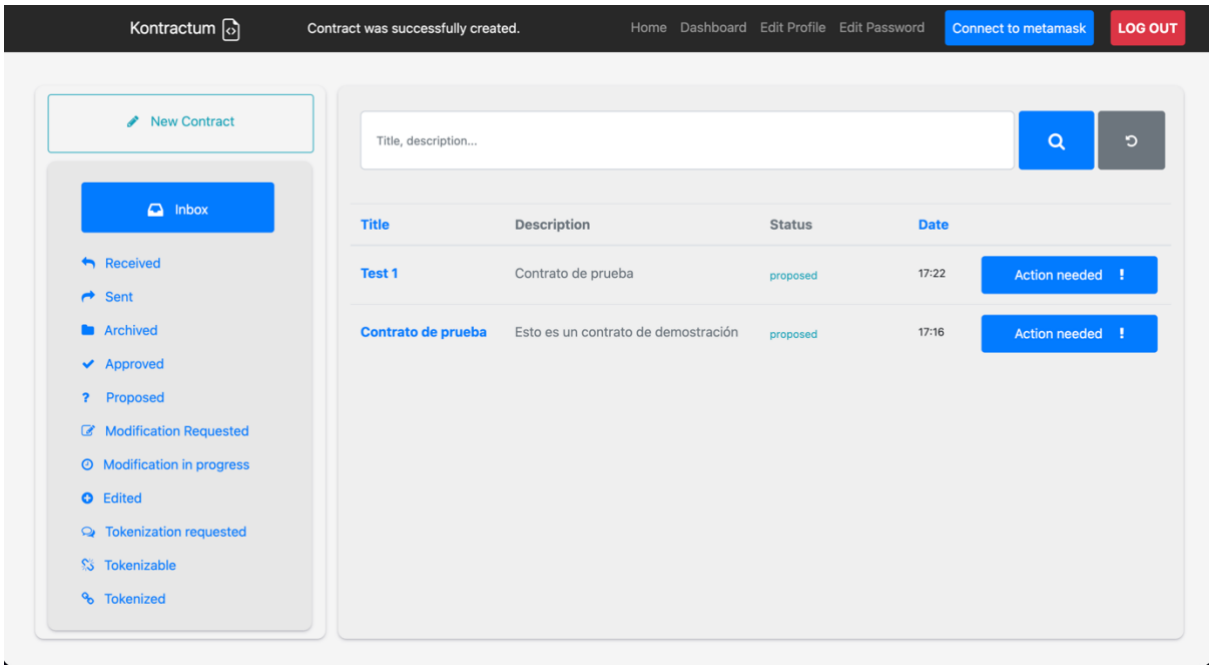

Figura 68 - Notificación al crear contrato

Al pulsar el botón de acción requerida, nos dirige a una nueva vista, los detalles y acciones del contrato, en este caso, como el usuario es beneficiario del contrato, tenemos la acción de aceptar o

rechazar la propuesta de contrato. El resto de los campos de los detalles del contrato están desactivados, solo podemos hacer click en “Open Contract”, para ver el documento ligado al contrato.

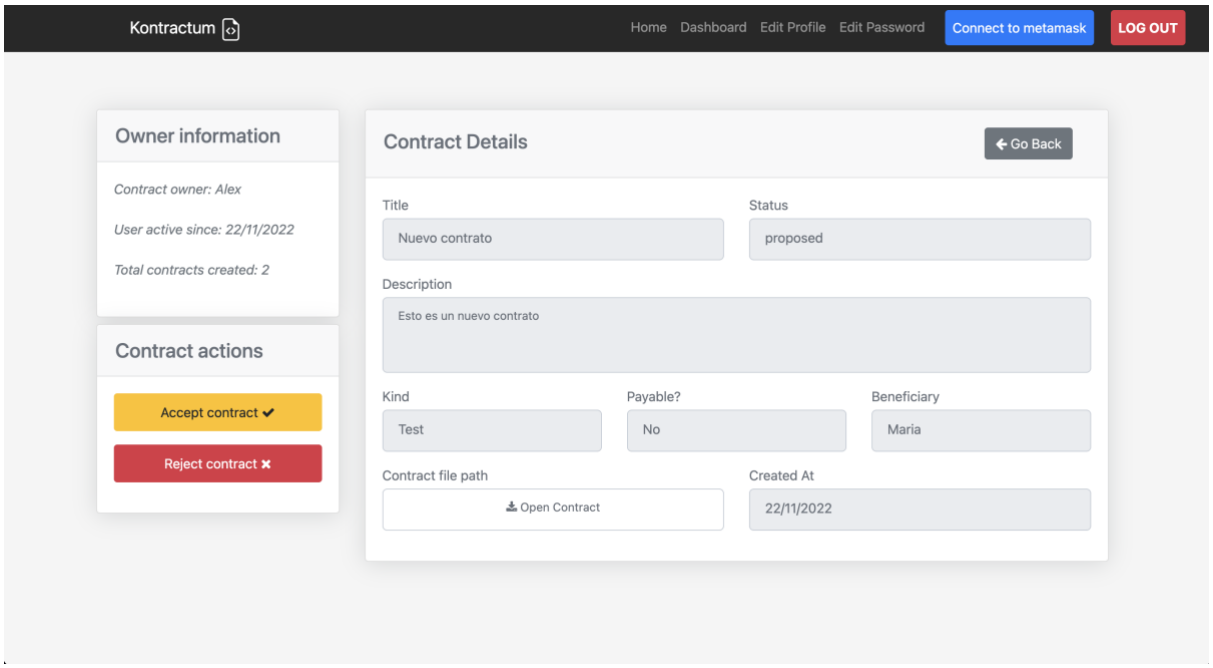

Figura 69 - Ver contrato y acciones

La aplicación también incorpora un dashboard para el usuario.

En la columna de la izquierda podemos ver información adicional del usuario, como el número de teléfono, una web o red social y su avatar. Además, podemos ver un breve resumen personal de la cantidad de contratos creados y en cuantos participa en total.

En la parte derecha del dashboard vemos una zona “About”, donde el usuario puede indicar una breve descripción personal. Los “User Badges” son distintivos del usuario que indican su rol dentro de la plataforma, así como los ciertos privilegios que el rol le otorga.

En la parte inferior podemos ver la actividad reciente del usuario, resúmenes cortos sobre los últimos 5 contratos.

El apartado Stats no está implementado, pero se explicará en Trabajos Futuros.

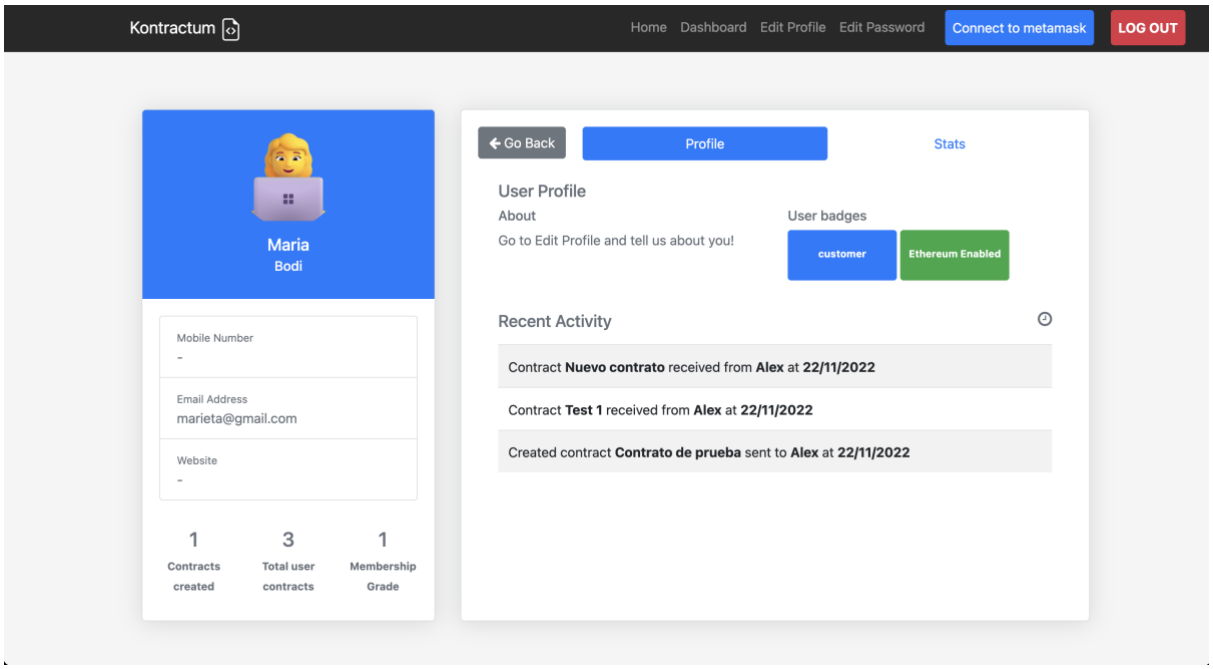

Figura 70 - Dashboard de usuario

Al hacer clic en “Edit Profile” se nos dirige a la siguiente vista, donde podemos actualizar nuestros datos personales, así como nuestra información adicional.

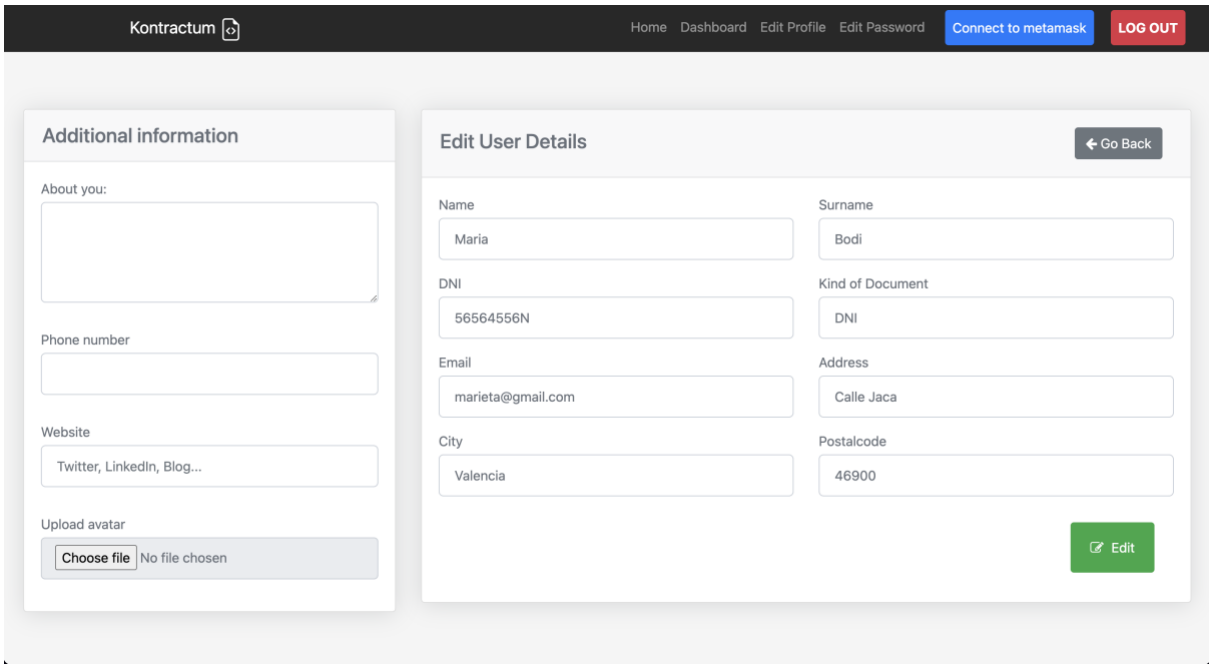

Figura 71 - Modificar datos personales

Al hacer click en “Edit Password” se nos dirige a la siguiente vista, donde podemos cambiar nuestra contraseña actual por una nueva y, si no estamos contentos con el servicio, borrar nuestra cuenta del sistema, así como nuestros datos personales.

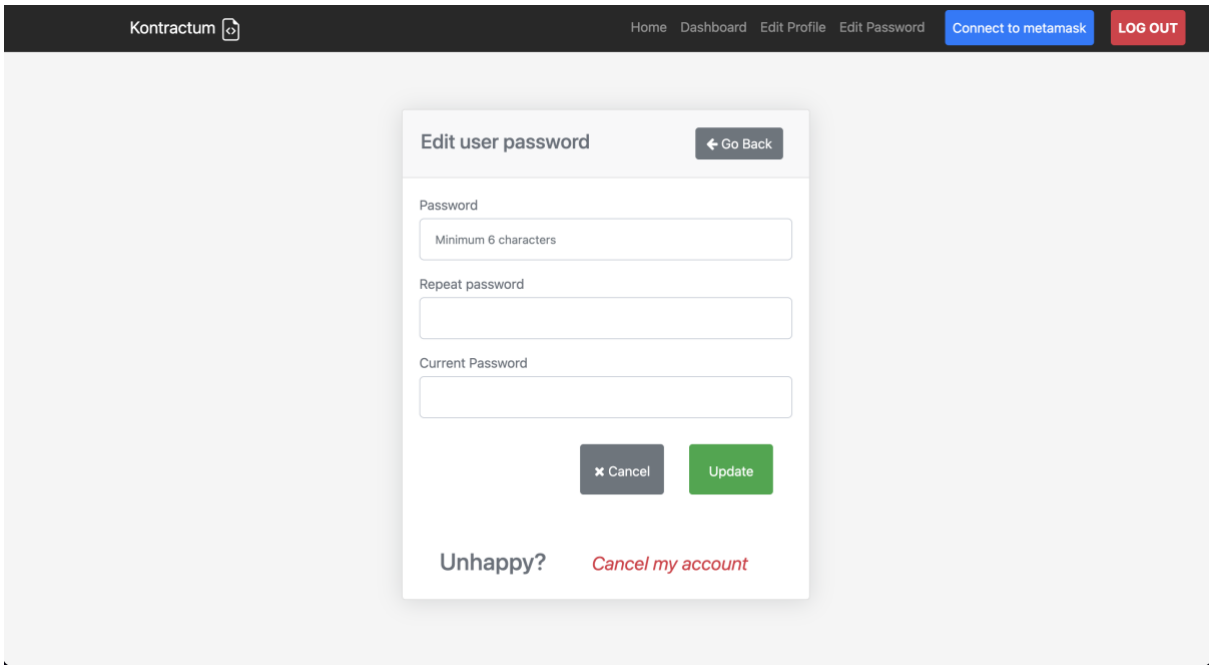

Figura 72 - Cambiar contraseña / Eliminar cuenta

4.2 Evaluación de la aplicación

Una vez desarrollada la aplicación web, se ha realizado una evaluación de la usabilidad de la aplicación. En este caso, puesto que ya se ha realizado una evaluación con usuarios sobre el prototipo de la aplicación, se utilizará la técnica evaluación heurística. Para ello, se utilizará como heurística los 10 principios de usabilidad de Jakob Nielsen.

A continuación, se evalúa el cumplimiento de los 10 principios de usabilidad de Nielsen en la aplicación desarrollada.

- Visibilidad del estado del sistema: El sistema (web, aplicación o cualquier otro producto digital) debe siempre mantener informado al usuario de lo que está ocurriendo.

- Parecido entre sistema y mundo real: El sitio web o aplicación tiene que emplear el lenguaje del usuario, con expresiones y palabras que le resulten familiares. Además, la información debe aparecer en un orden lógico y natural.

- Control y libertad del usuario: En caso de elegir alguna opción del sitio web por error, el usuario agradecerá disponer de una "salida de emergencia" para abandonar el estado no deseado en que se halla. Debe poder deshacer o repetir una acción previamente realizada.

- Consistencia y estándares: Es importante establecer convenciones lógicas y mantenerlas siempre. El usuario no tiene por qué saber que diferentes palabras, situaciones o acciones significan lo mismo.

- Prevención de errores: Se intenta guiar al usuario en los campos de inicio de sesión y registro para evitar que se cometan errores, estos campos contienen validadores de datos indicando si se ha producido algún error.

- Reconocer es mejor que recordar: Debemos hacer visibles acciones y opciones para que el usuario no tenga que recordar información entre distintas secciones o partes del sitio web o aplicación.

- Flexibilidad y eficiencia de uso: Los aceleradores o atajos, por ejemplo, pueden hacer más rápida la interacción para usuarios expertos, de tal forma que el sitio web o aplicación sea útil tanto para usuarios básicos como avanzados. Como un checkbox de recordar usuario.

- Diseño estético y minimalista: Las páginas no deben contener información innecesaria. Cada información extra compite con la información relevante y disminuye su visibilidad.

- Ayudar a los usuarios a solucionar los errores: Los mensajes de error se deben entregar en un lenguaje claro y simple, indicando en forma precisa el problema y sugerir una solución constructiva al problema.

- Ayuda y documentación: Aunque es mejor que el sitio web o aplicación pueda ser usado sin ayuda, puede ser necesario proveer cierto tipo de ayuda. En este caso, la ayuda debe ser fácil de localizar, especificar los pasos necesarios y no ser muy extensa.

La aplicación cumple 9 de los 10 principios de usabilidad de Jakob Nielsen, por tanto, podemos afirmar que la aplicación cumple con los principios de Nielsen.

4.3 Encuesta sobre interfaz gráfica, funcionalidad, y análisis de producto

Una vez finalizada la interfaz gráfica, así como las funcionalidades básicas de la aplicación, se ha realizado una encuesta abierta para conocer las opiniones sobre aspectos como la interfaz gráfica de usuario, la funcionalidad de la aplicación, usabilidad y enfoque de producto.

Para recibir feedback, se ha preparado un entorno gráfico en Figma con wireframes interconectados entre ellos simulando el flujo real de la aplicación.

Experiencia Kontractum

Las preguntas realizadas en la encuesta han sido separadas en 4 grupos.
- Cuestiones sobre diseño
- Cuestiones sobre funcionalidades
- Opciones de pagos
- Valoraciones finales

A continuación, se mostrarán las preguntas relacionadas con el diseño y sus correspondientes gráficas:

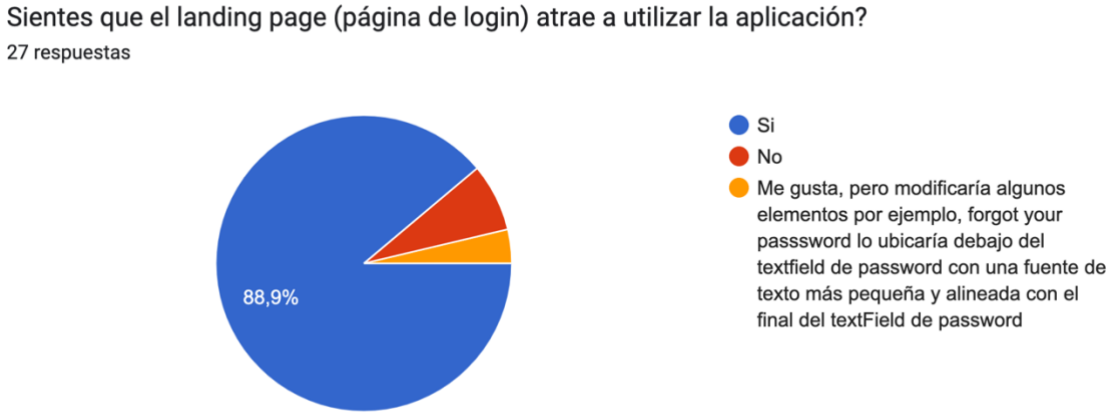

Figura 73 - Grafica 1 sobre diseño

Figura 74 - Gráfica 2 sobre diseño

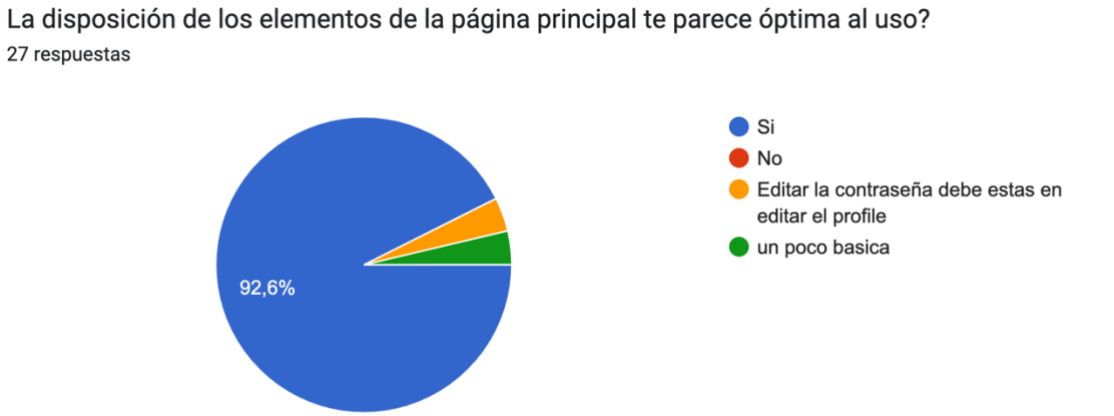

Figura 75 - Gráfica 3 sobre diseño

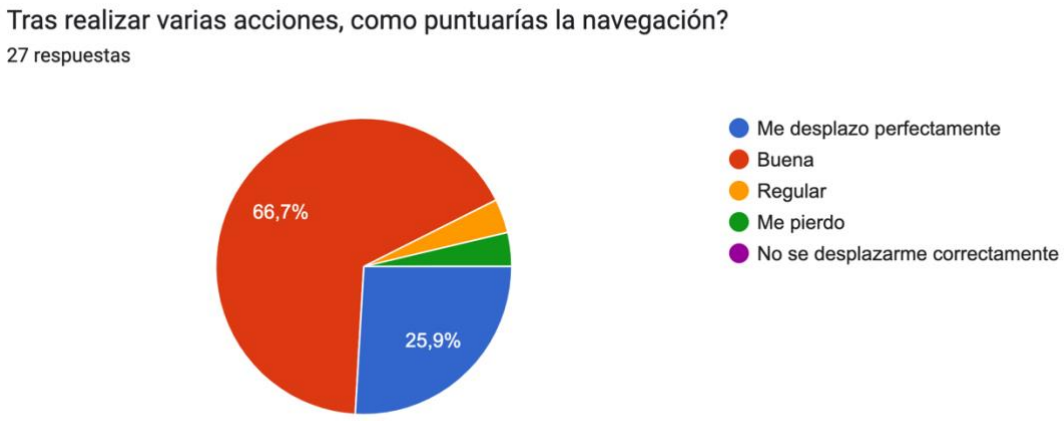

Figura 76 - Gráfica 4 sobre diseño

Figura 77 - Gráfica 5 sobre diseño

De las respuestas obtenidas podemos afirmar que:

- El landing page es agradable, aunque estaría bien crear un logotipo y mostrarlo en la bienvenida
- La disposición de los elementos es correcta y parece óptima al uso
- El desplazamiento por la web no es perfecto, pero es bueno
- El diseño obtiene una calificación notable alta

Ahora pasaremos a cuestiones sobre funcionalidad:

Figura 78 - Gráfica 1 sobre funcionalidad

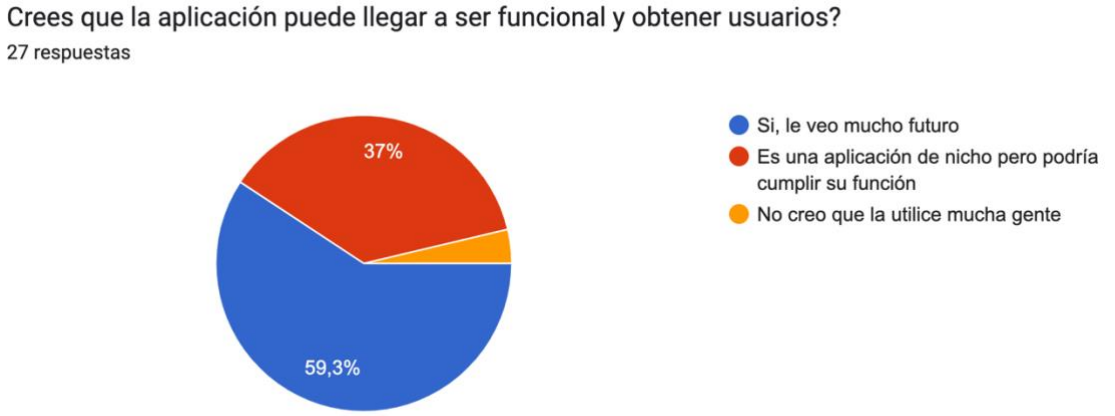

Figura 79 - Gráfica 2 sobre funcionalidad

Figura 80 - Gráfica 3 sobre funcionalidad

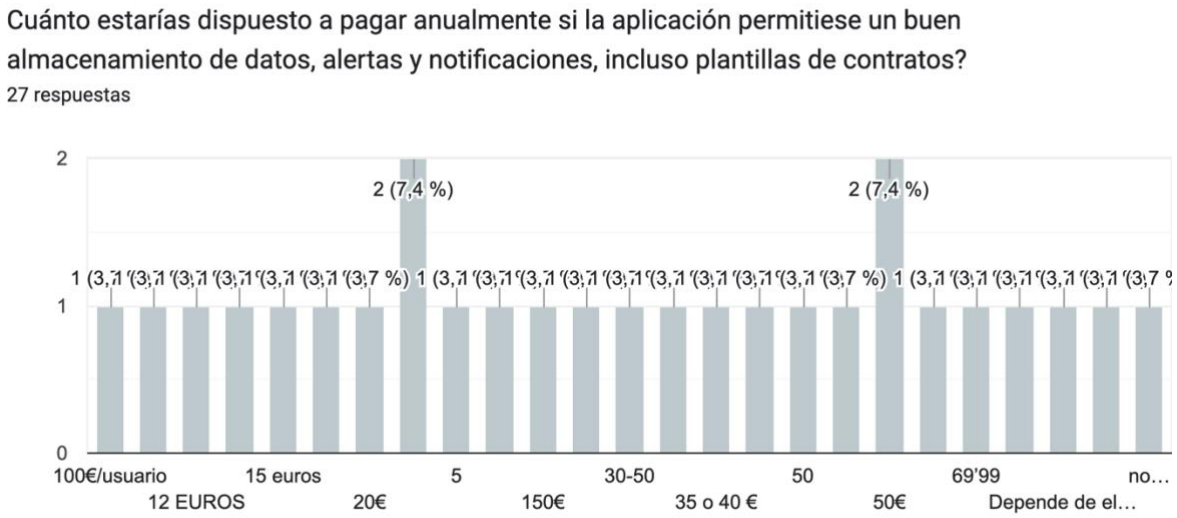

Figura 81 - Gráfica sobre precio de producto

Algunas de las conclusiones obtenidas tras las anteriores preguntas son:

- Las funcionalidades generales son correctas, aunque podrían añadirse o pulir algunas de las ya existentes.
- La mayoría de las respuestas sobre el futuro de la aplicación son positivas, podría captar usuarios y crear un nicho de mercado.
- La mayoría de las respuestas indican que la aplicación debería de ser gratuita, mientras que estarían dispuestos a pagar si se añadiesen ciertas funcionalidades extra.

La horquilla de precio a pagar varía entre los 5 y los 5000 euros anuales, siendo estos dos datos puntos atípicos en la gráfica. Ahora realizaremos el cálculo de la media y la mediana de los precios contando y sin contar, los datos atípicos.

Contando datos atípicos:

- Media: 289,45 €
- Mediana: 50 €

Sin contar datos atípicos:

- Media: 71,84 €
- Mediana: 45 €

Estos valores son meramente informativos y han servido al alumno para tener una idea de cómo enfocar la posible monetización de la aplicación.

Tras los resultados sobre el diseño, se ha planteado la creación de un logotipo/marca de la web para mostrarla tanto en el header como en la landing page. Las encuestas mostraban que la landing page era buena pero, un logotipo podía dar más personalidad y profesionalidad a la plataforma.

Por tanto, se ha creado y diseñado un logotipo, acorde a la estética de la web y con referencias a la blockchain.

Figura 82 - Logotipo 1 negro

Figura 83 - Logotipo 2 blanco

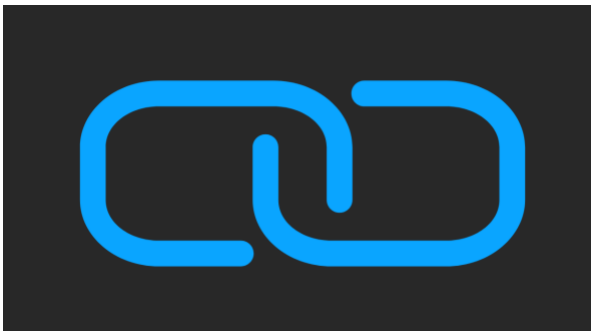

Figura 84 - Sólo logo negro

Figura 85 - Sólo logo blanco

Figura 86 - Landing page renovada

Quedando de tal forma la landing page, dándole un toque más profesional y con un branding renovado, moderno y minimalista.

5 Conclusiones

Kontraktum, la aplicación web enfocada a mejorar la gestión online de contratos, pretende ser un software que ayude a todas aquellas personas que traten con este tipo de documentos en su día a día.

Ha sido desarrollada con la intención de acercar las posibilidades que ofrecen los gestores de correos como Gmail a un nicho que no ha sido explotado lo suficiente, como es el campo de los contratos.

Seis meses de trabajo han dado lugar a la creación de una plataforma capaz de ofrecer un servicio que, después de analizar el mercado, ninguna otra plataforma llega a satisfacer.

Desarrollar en Ruby on Rails me ha hecho comprender muchas virtudes del patrón modelo vista controlador, así como trabajar con un lenguaje orientado a objetos y mejorar mis habilidades técnicas.

Por otro lado, la realización de este proyecto me ha ayudado a recordar y afianzar muchas de las enseñanzas que el grado en ingeniería informática me ha impartido durante 4 años. La mayoría de estos conocimientos existían, pero se encontraban inconexos y carecían de estructura, pero puedo afirmar que, en el momento de finalización del proyecto, estos se encuentran relacionados y unificados.

Además, he aprendido sobre desarrollo de producto, pensando en la aplicación como un servicio a prestar o “poner en venta”, teniendo en cuenta también que hay que aprender a sacar partido de nuestros conocimientos.

Todo esto me ha motivado para, en el futuro, seguir trabajando en este campo y continuar desarrollando software que pueda suponer una mejora para las personas, facilitándoles la vida u ofreciéndoles un valor no existente en el mercado.

5.1 Objetivos alcanzados

Estos son los objetivos específicos que hemos alcanzado:

- Desarrollo de un sistema informático que sirve como herramienta de gestión y control de contratos

- Análisis de requerimientos funcionales para un uso completo y correcto de la aplicación

- Desarrollo de un back-end capaz de encargarse de las peticiones realizadas por los usuarios de la aplicación

- Desarrollo de un front-end amigable, atractivo e intuitivo, que acompañe al usuario en todas sus acciones

- Realización de tests para las diferentes funciones de la aplicación, asegurando codificaciones de pruebas, desarrollo continuo y una correcta refactorización del código.

Como aspectos positivos del desarrollo del proyecto y tras experimentar todas las fases del proceso de creación, podemos resaltar los siguientes.

Por un lado, he podido aprender y profundizar en el uso de tecnologías que se emplean en las diferentes capas del desarrollo web, como el framework Rails, con su lenguaje nativo Ruby, utilizado para estructurar la parte trasera del proyecto, tanto como HTML y CSS, para maquetar y crear una interfaz atractiva, y GitHub, para llevar un control de versiones del proyecto.

Por otro lado, he profundizado en aspectos como desarrollo de producto, pensando en el proyecto como un negocio a explotar y adelantando posibles funcionalidades si la aplicación tiene buen recibimiento. También me he abierto al mundo del testing, he descubierto su potencial y sus ventajas, así como que una buena disciplina de control de tests puede llevarte a crear aplicaciones fuertes y robustas. Y por supuesto me he abierto al framework de Ruby on Rails, el cual me ha hecho sentirme muy acompañado durante el aprendizaje.

Sin embargo, no todo ha ido rodando y han aparecido algunos problemas durante el desarrollo. Como por ejemplo la introducción al testing. Al no haber creado nunca tests, la curva de aprendizaje de esta metodología no ha sido muy amigable, muchas veces por la poca experiencia, no entendía sobre qué debía de hacer tests y cómo debía de comprobar su buen comportamiento.

Realizar la integración del blockchain en el framework de Ruby, Ruby on Rails tampoco ha sido tarea fácil. Durante el proceso, han aparecido muchos problemas de compatibilidad, además no existen muchas plataformas Rails que incorporen blockchain, por lo tanto, no me he podido guiar por nadie. Pero el trabajo ha salido adelante.

5.2 Trabajos futuros

A pesar de que el proyecto final cumple con todo lo establecido junto al tutor del proyecto, el margen de mejora en cuanto a convertir la aplicación en un producto es bastante grande.

Considero que se pueden añadir muchas funcionalidades, como:

- Un chatbot que te ayude en alguna parte del proceso
- Alojar la aplicación en un servidor web real, mejorando así la seguridad y accesibilidad de todos los usuarios
- Añadir pasarela de pago para añadir ciertas funcionalidades al usuario
- Implementar una serie de estadísticas que permitan al usuario tener más información sobre el uso de la plataforma.

El diseño también puede dar más de sí. El aspecto de la web a mi parecer es agradable, minimalista y sobrio, pero con un poco más de tiempo y conocimientos, se podría obtener algo más atractivo e interesante.

Convertir el proyecto en un producto también ha sido un pensamiento recurrente en las etapas finales de la aplicación, una vez todas las funcionalidades habían sido implementadas y la interfaz gráfica también acompañaba, por lo tanto, sería un hito a considerar.

De todas formas, estoy muy contento con el esfuerzo realizado y el trabajo logrado.

6 Bibliografía

[1] N. Szabo, «Formalizing and Securing Relationships on Public Networks,» First Monday, 2(9). , [En línea]. Available: https://doi.org/10.5210/fm.v2i9.548.

[2] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» 2008. [En línea]. Available: https://bitcoin.org/bitcoin.pdf.

[3] T. Ohno y N. Bodek, Toyota Production System: Beyond Large-Scale Production, Japan: Productivity Press, 1988.

[4] D. Astels, Test Driven development: A Practical Guide, 2003.

[5] W. Clark, W. N. Polakov y F. W. Trabold, The Gantt chart : a working tool of management, New York: Ronald Press, 1923.

[6] A. Berson, Client-server architecture, New York: McGraw-Hill, 1992.

[7] D. H. Hansson y R. C. Team, «Ruby on Rails,» 2005. [En línea]. Available: https://rubyonrails.org/.

[8] J. A. Muro, «Qué es un ORM?,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html.

[9] Railsguides, «Active Record Basics,» [En línea]. Available: https://railsguides.es/active_record_basics.html.

[10] «Qué es HTML,» 14 de Agosto de 2021. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-html.html.

[11] Y. Matsumoto, 1995. [En línea]. Available: https://www.ruby-lang.org/.

[12] M. contributors, «What is CSS?,» 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS.

[13] R. Content., «Bootstrap: guía para principiantes de qué es, por qué y cómo usarlo.,» 2021. [En línea]. Available: https://rockcontent.com/es/blog/bootstrap/.

[14] M. contributors, «JavaScript,» 2022. [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

[15] «Solidity Documentation.,» [En línea]. Available: https://solidity.readthedocs.io .

[16] «Ethereum,» [En línea]. Available: https://ethereum.org/es/.

[17] T. Suite, «Ganache ONE CLICK BLOCKCHAIN,» [En línea]. Available: https://trufflesuite.com/ganache/.

[18] b. academy, «What are dApps?,» [En línea]. Available: https://academy.bit2me.com/en/what-are-dapps/.

[19] A. bit2me, «¿Qué es MetaMask? La forma más fácil de usar dApps,» [En línea]. Available: https://academy.bit2me.com/que-es-metamask-la-forma-mas-facil-de-usar-dapps/.

[20] R. D. Hipp, «SQLite,» 2020. [En línea]. Available: https://www.sqlite.org/index.html.

[21] J. Aguilar, «¿Qué es el patrón MVC en programación y por qué es útil?,» 2019. [En línea]. Available: ¿Qué es el patrón MVC en programación y por qué es útil?.

[22] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design patterns: elements of reusable object-oriented software, Boston: Addison-Wesley Longman Publishing Co, 1995.

[23] B. &. C. K. Still, Fundamentals of User-Centered Design: A Practical Approach, CRC Press, 2016.