UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Traspaso inmediato, inteligente, verificable y seguro de títulos de propiedad usando *Blockchain* y *NFTs*

Trabajo de graduación en modalidad de Megaproyecto Tecnológico presentado por
Andy Samuel de Jesús Castillo Barrientos
Marco José Fuentes Lima
Gian Luca Rivera Biagioni
Francisco Guillermo Rosal Ortega

para optar al grado académico de Licenciados en
Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala,
2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Traspaso inmediato, inteligente, verificable y seguro de títulos de propiedad usando *Blockchain* y *NFTs*

Trabajo de graduación en modalidad de Megaproyecto Tecnológico presentado por
Andy Samuel de Jesús Castillo Barrientos
Marco José Fuentes Lima
Gian Luca Rivera Biagioni
Francisco Guillermo Rosal Ortega

para optar al grado académico de Licenciados en
Ingeniería en Ciencia de la Computación y Tecnologías de la Información

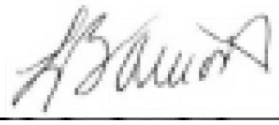
Guatemala,
2022

Vo.Bo.:

(f) _____

Ing. Roberto Chiroy

Tribunal Examinador:



(f) _____

MSc. Douglas Barrios

Fecha de aprobación: Guatemala, 02 de diciembre de 2022.

Dedicatoria

Dedico este trabajo a mi padres, Cluadia Barrientos y Álvaro Castillo, quienes durante todo mi vida siempre han estado a mi lado y me han demostrado su amor. Me han apoyado en los momentos duros y buenos y siempre me motivan a mejorar y dar lo mejor de mi. Gracias por todos los sacrificios que hicieron para siempre brindarme las mejores oportunidades, son un gran ejemplo como padres.

También, se lo quiero dedicar a mi hermano, Jose Daniel Castillo, una de las personas más especiales en mi vida. Gracias por acompañarme durante todos estos años y seguiré intentando ser el mejor ejemplo y apoyo durante toda tu vida.

Por último, quiero dedicar el trabajo a todos mis amigos con los que conviví todos estos años. Por todas las risas, desvelos, momentos buenos y malos que pasamos juntos, sin todos ustedes no hubiera sido lo mismo.

Andy Samuel Castillo Barrientos

Dedicatoria

A mi famila, especialmente a mis padres Marco Fuentes y Janina Lima, mi hermano, tíos, primos y abuelos, quienes con su amor, trabajo y esfuerzo me enseñaron incontables lecciones de vida sin ninguna de las cuales sería posible haber llegado a este momento.

A mi novia, quien nunca dudó de mí.

A mis amigos, quienes a través de su tiempo y paciencia me ayudaron a mantener la frente en alto en los momentos más difíciles.

A todos aquellos que formaron parte de este viaje académico que hace 20 años daba su comienzo y hoy culmina.

Marco Fuentes

Dedicatoria

A mi familia, en especial a mis padres, Marzia Biagioni y Miguel Rivera, por estar presente en cada momento de mi vida, por apoyarme en todos mis sueños y sobre todo en mis locuras, por ser el ejemplo perfecto de esfuerzo y sacrificio que me inspiran cada día y en especial por creer en mi y hacerme sentir orgulloso de tenerlos como padres. Sin ustedes nada de esto sería posible, gracias por su apoyo incondicional y gracias por ser lo mejor que tengo en la vida.

A mis amigos, quienes me acompañaron en este viaje lleno de retos y de grandes satisfacciones. A ustedes que me permitieron vivir una de las mejores experiencias de mi vida gracias a su apoyo y las historias vividas juntos. En especial les agradezco a ustedes, Francisco Rosal, Andy Castillo y Marco Fuentes, por ser parte de este proyecto y sobre todo por ser como mis hermanos.

A Roberto Chiroy, quien con su dedicación y paciencia me transmitió la pasión por la tecnología blockchain.

Por último, a todas aquellas personas que han contribuido a mi crecimiento personal y profesional.

Gian Luca Rivera Biagioni

Dedicatoria

Dedico este trabajo a mis padres, Griselda y Federico, quienes siempre han querido para mí más de lo que podían haber imaginado y dieron tanto de sí mismos. Gracias por todos los sacrificios que hicieron, por forjarme en la persona que soy y siempre apoyarme y motivarme para lograr mis metas y sueños. Seguiré esforzándome para llevar en alto el nombre que me dieron. Muchos de mis logros se los debo a ustedes, en especial este.

Adicionalmente, dedico este trabajo a mis hermanos Emanuel, David y Pablo, por estar en las buenas, en las malas y por todos los recuerdos que tenemos desde que nacieron y llegaron a mi vida. Seguiré esforzándome por ser un buen ejemplo para ustedes y poder ayudarlos siempre.

También se lo dedico a los que me han dado oportunidades para demostrar mis capacidades y superarme.

Además, agradezco a Samuel Chávez, quien desde el colegio fomentó la pasión que tengo por la programación. También agradezco a Carlota Marroquín, Israel Hernández, Eddy Petz y Luis Donis por su apoyo, las oportunidades que me han dado y el conocimiento que me han transmitido.

Agradezco a Roberto Chiroy por ser mi asesor y guiarme en la elaboración de este trabajo. Y especial agradecimiento a mis compañeros y amigos Gian Luca Rivera, Andy Castillo y Marco Fuentes; con quienes he compartido estos años en la universidad y quienes trabajaron junto a mí este proyecto.

Doy gracias a Dios por darme la familia que siempre ha estado a mi lado y por los amigos que siempre han creído en mí. Les agradezco a todos por animarme a ser mejor cada día y al sacrificio que me ha llevado a este gran logro.

Gracias,

Francisco G. Rosal Ortega

Prefacio

Este trabajo de graduación consta de la unión de cuatro módulos independientes que presentan una propuesta de innovación tecnológica en registro de títulos de propiedad usando tecnología *blockchain*. En este proyecto se abordan temas de inteligencia artificial, redes convolucionales, reconocimiento de imágenes, algoritmos de arboles de decisión, medidas y protocolos de seguridad de datos, análisis de riesgo, políticas de seguridad, cifrado de datos, controles de acceso, KYC, JWT, *blockchain*, *smart contracts*, *NFTs*, *Web3*, *UI/UX*, administración de proyectos, ambientes de desarrollo, *testing* automático, almacenamiento en la nube, *DNS*, desarrollo *backend*, base de datos, *APIs*, infraestructura de nube y *CI/CD*, entre otros. Finalmente, se detallan algunas conclusiones y recomendaciones a tomar en cuenta para posteriores implementaciones. El nombre del Megaproyecto, sus módulos y sus autores se detallan a continuación.

Traspaso inmediato, inteligente, verificable y seguro de títulos de propiedad usando *Blockchain* y *NFTs*

Módulos

Marco Fuentes Desarrollo e implementación de algoritmos de validación de datos de propiedades y sistemas de recomendaciones usando IA.

Andy Castillo Diseño e implementación de medidas y protocolos de seguridad en una plataforma para el traspaso de títulos de propiedad.

Gian Luca Rivera Diseño e implementación de la tecnología de *Blockchain* y *Web3* por medio de *smart contracts* y *NFTs*.

Francisco Rosal Diseño e implementación de infraestructura de nube, base de datos, servidor y *API* para integrar con aplicaciones *Blockchain* y *Web3*.

Índice

Listado de figuras

Listado de tablas

Listado de *scripts*

Resumen

El presente trabajo de graduación propone una transformación digital al proceso de transferencia de títulos de propiedad. El principal objetivo de este proyecto es automatizar los procesos relacionados a la compraventa de propiedades utilizando *blockchain* como tecnología principal para poder brindar trazabilidad y confiabilidad al proceso de transferencia de títulos de propiedad. Para hacer esto posible se desarrolló una plataforma web que almacena el registro tokenizado de las propiedades, y permite la transferencia de activos entre usuarios identificados.

Los resultados evidencian que la utilización de tecnologías como *blockchain*, *smart contracts* y *NFTs* son una propuesta viable, funcional y transparente para automatizar el proceso de traspaso de títulos de propiedad de manera verificable y rápida.

Abstract

This graduation project proposes a digital transformation to the process of transferring property titles. The main objective of this project is to automate the processes related to the purchase and sale of properties using *blockchain* as the main technology to provide traceability and reliability to the process of transferring property titles. To make this possible, a web platform was developed that stores the tokenized record of the properties and allows the transfer of assets between identified users.

The results show that the use of technologies such as *blockchain*, *smart contracts*, and *NFTs* is a viable, functional, and transparent proposal to automate the process of transferring property titles in a verifiable and fast way.

CAPÍTULO 1

Introducción

La industria inmobiliaria es un gigante en la economía mundial. Esta industria es la responsable de toda diligencia relacionada con bienes inmuebles. La frase "bienes raíces" se mencionó por primera vez en la década de 1660 y se convirtió en una frase legal para referirse a una concesión real de tierras. Los bienes son activos fijos que no se pueden convertir en efectivo, pero generan ingresos y plusvalía con el paso del tiempo. [81]

Esta industria es de las más grandes y presentes en todo el mundo, probablemente es de las industrias más antiguas, pero es una de las menos revolucionarias en cuanto a transformación digital. Hablando de las características de cualquier diligencia relacionada y respecto a bienes raíces, se puede mencionar que es lento y poco eficiente. Esto se debe a que los procesos de adquisición de propiedades se siguen realizando igual que hace muchos años. Generalmente, se debe conseguir la papelería, se debe agendar numerosas citas, visitas, contratación de abogados, asesores y otros intermediarios, etc... para cerrar un único trato. Es un negocio donde muchas personas entran en juego y esto puede elevar fácilmente la existencia de fraudes y estafas. Es incluso más complejo cuando es un interés internacional, ya que extranjeros pueden estar interesados en adquirir propiedades en diferentes países.

La inversión inmobiliaria se ha reconocido como una de las inversiones más seguras. ¿Por qué no asegurar la transacción de transferencia de títulos de las propiedades y sus procesos relacionados? Generalmente, las personas no tienen idea de la importancia de la protección de sus derechos sobre la tierra, y es importante resaltar que estos derechos son fundamentales para estimular la inversión y el crecimiento. [11]

Actualmente, la tecnología está tomando un papel muy importante en los aspectos de automatización en diferentes industrias. Sin embargo, hay algunas en las que todavía hay una gran resistencia y muchos procesos todavía se gestionan de forma tradicional. Un ejemplo de esto es la industria de bienes raíces. La compra de bienes inmuebles es un proceso lento y, a menudo, desorganizado que depende de múltiples intermediarios y una comunicación retardada. Existe una falta de plataformas disponibles para la transferencia y registro electrónico de bienes inmuebles. La automatización de estos procesos podría generar grandes ganancias, tanto para la industria como para el resto de los involucrados. Con la ayuda de la tecnología *blockchain*, esto es posible utilizando diferentes protocolos como *Smart Contracts* y *Non-Fungible Tokens (NFT)* donde la compraventa de una propiedad es transparente, trazable e inmutable.

Convertir una propiedad a *NFT* permite transferirla de forma segura de una billetera digital a otra y verificar de manera auténtica y automática el propietario del mismo. Esto se logra a través de *Smart Contracts*, donde toda la identidad de la propiedad necesaria para el proceso de compraventa está vinculado a un *NFT* de la propiedad. La importancia de esto radica en el debido registro de los derechos de propiedad. *Blockchain* se puede implementar en la industria de bienes raíces gracias a la tendencia actual de tokenizar activos no fungibles. Los tokens no fungibles (*NFT*) nos ayudarán a lograr la actualización más importante en la industria de bienes raíces; el traspaso de propiedades debe ser inmediato, inteligente, verificable y seguro.

Los contratos, las transacciones y los registros de los mismos se encuentran entre las estructuras definitorias de los sistemas económicos, legales y políticos. Protegen los activos y establecen límites organizacionales. Establecen y verifican identidades y narran eventos. Gobiernan las interacciones entre naciones, organizaciones, comunidades e individuos. Orientan la acción empresarial y social. Y, sin embargo, estas herramientas críticas y las burocracias formadas para administrarlas no se han mantenido al día con la transformación digital de la economía [40].

Esto se puede lograr por medio de una plataforma en línea que permita investigar, pagar, registrar y transferir de manera confiable y segura la propiedad de un bien inmueble. Por medio de la tecnología de *blockchain* y *NFTs* se puede tener un registro lo mas transparente posible para que la transferencia de propiedad sea casi inmediata a la hora de realizar la compra.

CAPÍTULO 2

Antecedentes

En cuanto a proyectos relacionados; 20Mission[78] fue una comunidad de 41 habitaciones en San Francisco llena de creadores de arte y tecnología en el distrito Mission. 20Mission arrendaba sus espacios a través de *NFTs*. Los *NFTs* vendidos proporcionaban los derechos a un contrato de arrendamiento de 75 años a $1 mensual con cero tarifas de condominio, cero impuestos a la propiedad y cero utilidades. Es decir que todos los costos estaban incluidos. El *NFT* representaba el arrendamiento de un espacio físico del cual se tenía el 100 % de control sobre la habitación durante 75 años. El contrato de arrendamiento que se obtenía cumplía con los estándares legales requeridos.

Actualmente, puede encontrarse un proyecto similar en Estados Unidos llamado Propy [86]. Se trata de una plataforma en la que se vendió la primera propiedad *NFT* del mundo. El apartamento era propiedad de Michael Arrington, fundador de TechCrunch y Arrington XRP capital, y decidió vender su propiedad como un *NFT* para mostrar el poder de la tecnología *blockchain*, demostrando la manera en la que se puede innovar la industria inmobiliaria. Para vender el apartamento como un *NFT*, el proceso funcionó de manera ligeramente diferente a una venta de bienes raíces tradicional. El registro de la propiedad del apartamento se encontraba en Ucrania (debido a que el apartamento se ubicaba en este país) como una LLC estadounidense. Se subastó la propiedad y el ganador se convirtió en el propietario del *NFT* que otorgó los derechos legales a la LLC. Propy desarrolló los smart contracts y el marco legal adecuado para el mercado estadounidense y pronto lanzará una plataforma de subastas de *NFT*. Para esta subasta, Propy se asoció con Seen Haus para llevar a cabo la licitación y Helio Lending para asegurar el financiamiento a futuros propietarios de bienes raíces de *NFT*.

A nivel de tecnología este proyecto de Propy presenta una funcionalidad similar a la representada en este trabajo. Sin embargo, en el momento de escritura de este trabajo, se desconoce la infraestructura completa que usa esta aplicación. En su página web hablan únicamente de la utilización de tecnología *Web3* y *NFTs*. Propy representa una prueba piloto del concepto de un sistema de registro de propiedad basado en *blockchain* como el descrito en este trabajo.

CAPÍTULO 3

Justificación

Históricamente, la inversión inmobiliaria se ha reconocido como una de las inversiones más seguras, de manera que muchas personas aspiran a realizar este tipo de inversión, no solo como vivienda sino como método de capitalización. Entonces, ¿por qué no asegurar las propiedades y sus procesos relacionados? Generalmente, las personas no tienen idea de la importancia de la protección de sus derechos sobre la tierra. Lo que no saben es que, según el Banco Mundial, en el mundo solo el 30 % de la población tiene un título de propiedad debidamente registrado de sus tierras. Y es importante resaltar que los derechos sobre la tierra son fundamentales para estimular la inversión y el crecimiento, e incluso para reducir la pobreza. [11]

¿Cómo ayudar a resolver el problema? ¿Dónde queda la tecnología? En una encuesta realizada por PwC (PricewaterhouseCoopers) se encontró que la tecnología no es lo primero en la agenda de los CEOs de la industria inmobiliaria a nivel mundial. La mayoría de estos CEOs no ven la tecnología ni como amenaza, ni como oportunidad. [20]

Actualmente, *Blockchain* se considera como una de las tecnologías más seguras para la transferencia y almacenamiento de datos debido a sus características intrínsecas por la definición de su naturaleza. La transacción segura, confiabilidad y no repudio que provee el *Blockchain* por definición, permitirá un manejo más adecuado de los derechos sobre los registros de propiedad, que es un gran problema actual.

Blockchain, almacenamiento de datos descentralizado, público, inmutable y seguro. Es una base de datos que ofrece tolerancia a fallos, robustez, trazabilidad y transparencia. En una red de *Blockchain* el contenido es verificado por cada participante de la red (nodo) por medio de un mecanismo de consenso previamente definido. *Blockchain* se basa en *DLT (Distributed Ledger Technology)* lo que permite validar las transacciones que suceden dentro de la red y la trazabilidad de estas, porque cada nueva transacción está criptográficamente vinculada a la anterior formando una especie de cadena o historial. Esto hace que los datos sean prácticamente inmutables y trazables, es fácil detectar cualquier intento de alteración y evitar fraudes.

Con *blockchain* es posible imaginar un mundo en el que los contratos estén integrados en un código digital y almacenados en bases de datos transparentes y compartidas, donde estén protegidos

contra eliminación, censura y alteración. Cada proceso, cada tarea y cada pago tendrían un registro y una firma digitales que podrían identificarse, validarse, almacenarse y compartirse. Presentando la posibilidad de reducir la cantidad de actores en los procesos relacionados. Individuos, organizaciones, máquinas y algoritmos realizarían transacciones e interactuarían libremente entre sí con poca fricción.[40]

Sin embargo, al trasladar el proceso a un espacio virtual, las posibilidades de fraudes se multiplican, y ya en industrias como la de seguros médicos se han reportado millonarias pérdidas gracias a registros fraudulentos [17].

Es por esto que el producto se ve en la necesidad de contar con herramientas que ayuden a los clientes a identificar registros potencialmente fraudulentos antes de proceder con la compra. Usar asesores económicos individuales sería altamente costoso en términos de precio y tiempo, por lo que desarrollar una IA que se encargue de hacer este trabajo es una solución ideal. Además, esta IA presenta la ventaja de que puede ser entrenada con datos actualizados anualmente, de manera que siempre pueda predecir precios actualizados de propiedades y tomar en cuenta burbujas inmobiliarias, subidas o bajadas repentinas en precios en el mercado, etc.

Por otro lado, este traslado también trae otro reto: el de optimización de recursos que serán almacenados en el *smart contract* o bien en una base de datos externa. Si bien es útil que cada *NFT* correspondiente a una propiedad tenga imágenes descriptivas, se busca que cada imagen almacenada corresponda únicamente a espacios relevantes de la propiedad, tales como habitaciones, cocina, sanitarios, dormitorios o salas. De lo contrario, almacenar tal volumen de datos elevaría el costo de mantenimiento de la red y de cada transacción de *NFTs*. De nuevo, delegar esta tarea de validación y clasificación de imágenes a agentes humanos sería costoso y poco eficiente, razón por la cual un agente de IA que la realice sería más beneficioso. Además, en la actualidad se cuentan con algoritmos de este tipo en aplicaciones de fotografías (tales como Google Photos, o Fotos en iPhone), en donde un modelo de inteligencia artificial se encarga de analizar imágenes y clasificarlas por contenido, ubicación, personas o cosas que se encuentren presentes. Incluso algunos reconocen imágenes duplicadas en la galería y recomiendan al usuario eliminarlas para optimizar el espacio en el almacenamiento.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Utilizando Inteligencia Artificial y considerando la seguridad y autenticidad de los datos, automatizar el proceso de transferencia y pertenencia de derechos de propiedad utilizando *Blockchain*, por medio del almacenamiento de registros de propiedades representadas como *NFTs*.

4.2. Objetivos específicos

- Diseñar y desarrollar un *smart contract* que automatice el proceso de transferencia de títulos de propiedad por medio de la creación de un *NFT*.
- Mantener la transparencia y trazabilidad en todas las operaciones dentro la red de *blockchain*.
- Desarrollar *NFTs* con metadatos en donde los títulos de propiedad estén vinculados al mismo para que la transferencia sea una transferencia de propiedad física y virtual.
- Utilizar *Web3* para poder hacer la comunicación entre una red de *blockchain* y una interfaz web para mostrar los datos almacenados en la misma.
- Realizar transferencias de *NFTs* de una billetera digital a otra para reflejar la compra y venta de propiedades.
- Diseñar e implementar una infraestructura en la nube segura, para montar un servidor web.
- Diseñar y desarrollar una infraestructura *Backend*, que funcione como servidor e intermediario entre el *Frontend* y los datos almacenados en *Blockchain*.
- Crear un oráculo que exista dentro del servidor, que consulte y modifique, de manera segura, los datos almacenados en el *Blockchain* y los pueda procesar para su uso y manipulación sencilla.
- Desarrollar un sistema de verificación de datos que sea capaz de identificar si distintas fotografías pertenecen a ambientes principales de una propiedad inmobiliaria, tales como salas, sanitarios, habitaciones, cocinas o comedores.

- Implementar un sistema de recomendaciones de precios para potenciales clientes de propiedades.
- Implementar un *API* web que permita el acceso a dichos servicios de verificación y recomendaciones.
- Definir e implementar políticas de privacidad y métodos de autenticación seguros para la plataforma.
- Definir e implementar métodos de criptografía para el almacenamiento y transferencia de datos.
- Realizar un análisis de riesgos para detectar posibles vulnerabilidades y poder reducirlas o resolverlas.

CAPÍTULO 5

Alcance

Este Megaproyecto "**Traspaso inmediato, inteligente, verificable y seguro de títulos de propiedad usando *Blockchain* y *NFTs***"se desarrolló con el propósito de implementar una solución a la falta de transformación digital en la industria inmobiliaria. Atacando principalmente a lo concerniente a la transferencia de títulos de propiedad.

El concepto desarrollado en este proyecto esta limitado al registro de bienes inmuebles; sin embargo, el proyecto puede ser adaptado para que también sea aplicable a otros ámbitos de registros de propiedad de otros tipos de bienes e incluso de identidad con algunos cambios.

CAPÍTULO 6

Marco teórico

6.1. *Blockchain*

6.1.1. Definición

Blockchain es un libro mayor digital distribuido e inmutable para registrar transacciones de cualquier tipo, rastrear activos y generar confianza. Puede registrar información sobre transacciones de criptomonedas, propiedad de *NFT* o contratos inteligentes. Básicamente, cualquier cosa identificable puede ser rastreada y comercializada en una red de *blockchain*, reduciendo el riesgo y los costos para los involucrados. Un activo en una red de *blockchain* puede ser tangible, como una casa o un auto, o intangible, como patentes o marcas [48].

El libro digital se describe como una cadena que se compone por bloques individuales de datos. A medida que se añaden datos nuevos periódicamente a la red, se crea un nuevo bloque y se adjunta a la cadena. Esto implica que todos los nodos actualicen su versión del libro mayor *blockchain* para que sean idénticos [21].

Blockchain es ideal para obtener, almacenar y trasmitir información de forma segura. Al trabajar sobre la idea de un libro mayor distribuido, se puede almacenar datos compartidos, transparentes e inmutables, lo cual provee confiabilidad entre los participantes.

En una red *blockchain*, todos los participantes de la red tienen acceso al libro mayor distribuido y a su registro inmutable de transacciones. Además, ningún participante puede alterar o falsificar una transacción una vez grabada en el *blockchain*. Se le puede agregar funcionalidad a una red de *blockchain* por medio de los *smart contracts*. Estos *smart contracts* son almacenados en *blockchain* y constan de un conjunto de reglas que pueden ejecutarse bajo demanda o automáticamente y ofrecen diferentes formas de interactuar con *blockchain* [2].

La manera en que funciona una red de *blockchain* cuando se quiere crear un nuevo bloque es que la mayoría de los nodos de la red deben de verificar y confirmar la legitimidad de los nuevos datos antes de que se pueda añadir un nuevo bloque al libro mayor, cosa que es diferente en una base

de datos tradicional ya que una persona puede realizar cambios sin supervisión. Una vez los nodos validadores llegan a un consenso, el bloque se añade a la cadena y las transacciones subyacentes se registran en el libro mayor distribuido. Todas las transacciones están protegidas por medio de criptografía de manera que cada nodo necesita resolver ecuaciones matemáticas complejas para procesar una transacción [21].

Al ser *blockchain* un libro mayor, la forma en la que se trabaja es por medio de transacciones. Las transacciones muestran el movimiento de un activo y a medida que se producen más transacciones se registran en un bloque de datos. El bloque de datos puede almacenar: quien, que, cuando, cuanto e información adicional relevante. Cada bloque está conectado al anterior y al posterior, de modo que se forma una cadena de bloques que conforman el historial de transacciones. Por otro lado, podemos entender *blockchain* como una base de datos.

6.1.2. Ventajas

- No hay necesidad de intermediarios: usando *blockchain*, dos partes de una transacción pueden confirmar y completar algo sin trabajar a través de un tercero. Esto ahorra tiempo, así como el costo de pagar por un intermediario.
- Precisión en las transacciones: debido a que una transacción en una red *blockchain* debe ser verificada por varios nodos, esto puede reducir el error. Si un nodo tiene un error en sus registros, los temas nodos verían que los datos de ese nodo son diferentes y de esta manera detectar el error.
- Seguridad: una red descentralizada hace casi imposible que alguien realice transacciones fraudulentas ya que para entrar transacciones falsificadas se tendrían que comprometer cada nodo de la red y cambiar cada libro mayor.
- Transferencias eficientes: debido a que las redes *blockchain* funcionan las 24 horas del día, los 7 días de la semana, las personas pueden realizar transferencias (transacciones) financieras y de activos de manera más eficiente, especialmente a nivel internacional ya que no se necesita la confirmación manual de ninguna entidad [21].

6.1.3. Limitaciones

- Límite de transacciones por segundo: dado que una red *blockchain* depende de una red más grande para aprobar transacciones, hay un límite en la rapidez con la que se puede realizar transacciones y esto depende de la red de *blockchain*. En una red *blockchain*, la escalabilidad es un desafío.
- Altos costos energéticos: tener todos los nodos trabajando para verificar las transacciones requiere mucha más electricidad que una sola base de datos, de manera que crea una gran carga de carbono para el medio ambiente.
- Potencial de actividades ilegales: la descentralización de *blockchain* añade más privacidad y confidencialidad, lo que la convierte en una tecnología atractiva para los delincuentes y es mas difícil rastrear las transacciones ilícitas en *blockchain* [21].

6.2. Tipos de redes *blockchain*

6.2.1. Públicas

Son de naturaleza sin permiso, permiten que cualquiera se una y están completamente descentralizadas. Las *blockchain* públicas permiten que todos los nodos de la cadena de bloques tengan los mismos derechos para acceder a la red *blockchain*, crear nuevos bloques de datos y validar bloques de datos. Generalmente en este tipo de *blockchain* existen los mineros, quienes son encargados de crear bloques para las transacciones solicitadas en la red, resolviendo ecuaciones criptográficas. A cambio de este trabajo, los mineros ganan un porcentaje de la tarifa de transacción [59].

6.2.2. Privadas

Estas son redes de *blockchain* autorizadas y controladas por una sola organización en donde la autoridad central determina quién puede ser un nodo. También, la autoridad central decide qué permisos otorgar a cada nodo, ya que no necesariamente todos los nodos tienen los mismos derechos para realizar funciones dentro de la red. Estas son redes parcialmente descentralizadas debido a que el acceso público a la red de *blockchain* está restringido [59].

6.2.3. Consorcio

Son *blockchain* gobernadas por un grupo de organizaciones en lugar de una sola entidad, de manera que se disfruta más la descentralización con respecto a las redes privadas, de manera que son consideradas redes más seguras. Este tipo de redes requiere cooperación entre una serie de organizaciones [59].

6.2.4. Híbridas

Es una combinación de las redes de *blockchain* privadas y públicas ya que utiliza las características de ambos tipos de *blockchain*. Es controlada por una sola organización, pero con un nivel de supervisión realizado por la *blockchain* pública que es necesaria para realizar ciertas validaciones de transacciones. Es decir, el acceso a los recursos de la red es controlado por una o varias entidades, sin embargo, el libro de contabilidad es accesible de forma pública [59].

6.3. *Smart Contract*

Un contrato inteligente es un término utilizado para describir el código informático que ejecuta automáticamente todo o parte de un acuerdo y se almacena en una plataforma basada en *blockchain*. El código en sí se replica en varios nodos de una red *blockchain*, de manera que el contrato se beneficia de la seguridad, permanencia e inmutabilidad que ofrece la tecnología de *blockchain*. Antes de que el contrato se ejecute, se requiere un paso adicional, y es el pago de una tarifa de transacción para que el contrato se agregue al *blockchain* y se ejecute. Cuanto mas complejo sea el contrato

inteligente, más se debe de pagar de la tarifa de transacción [63].

Los contratos inteligentes funcionan siguiendo declaraciones de si/cuando, entonces (*if*, *else*) que están escritas en código en *blockchain*. Una red de nodos ejecuta las acciones cuando se han cumplido y verificado condiciones predeterminadas. El *blockchain* se actualiza cuando se completa la transacción de manera que la transacción no se puede cambiar y solo las partes a las que se ha concedido permiso pueden ver los resultados. Los beneficios de utilizar contratos inteligentes son muchos. Algunos de estos son:

- Velocidad, eficiencia y precisión: una vez que se cumple una condición, el contrato se ejecuta inmediatamente. Debido a que los contratos inteligentes son digitales y automatizados, no hay papeleo que procesar ni tiempo dedicado a conciliar errores que suceden menudo al rellenar documentos manualmente.
- Confianza y transparencia: debido a que no hay ningún tercero involucrado debido a que los registros cifrados de las transacciones se comparten entre los participantes, no hay necesidad de cuestionar si la información se ha alterado para beneficio personal.
- Seguridad: los registros de transacciones de *blockchain* están cifrados, lo que los hace muy difícil de comprometer. Además, debido a que cada registro está conectado a los registros anteriores y posteriores en un libro mayor distribuido (*blockchain*), los piratas informáticos tendrían que alterar toda la cadena para alterar un solo registro [46].

6.4. *NFT*

6.4.1. Fungibilidad

La fungibilidad se refiere a la intercambiabilidad de un activo. En términos económicos, la fungibilidad es la capacidad de de un activo para ser intercambiado con otros activos individuales del mismo tipo con el propósito de negociar valor. De tal manera, los activos fungibles implican el mismo valor. Las monedas fiduciarias o las criptomonedas, como Bitcoin y Ether, o incluso el oro, son activos fungibles ya que no existe una diferencia entre cada unidad de estas de tal manera que, por ejemplo, 1 Bitcoin = 1 Bitcoin o un billete de \$5 = otro billete de \$5. Por otro lado, los activos no fungibles, por definición, son únicos, no se pueden intercambiar y son irreemplazables. [26]

También existe un camino intermedio y es la semi-fungibilidad. Este, es un término relativamente nuevo y se refiere a la intercambiabilidad entre clases específicas de activos. Por ejemplo, las entradas para un concierto pueden ser intercambiadas si son para la misma función y en la misma localidad. Sin embargo, el tema interesante es que después de utilizar un activo semi fungible (como la entrada a un concierto), se convierte en una pieza única y a partir de entonces, es un activo no fungible. [42]

Figura 6.1: Activos fungibles, semi fungibles y no fungibles [34]

6.4.2. Tokenización

La tokenización se define como una forma nueva, más ágil, respaldada en un titulización basado en código y una estructuración basada en *blockchain* de tal manera que conecta un activo subyacente a un *smart contract*. Es el proceso mediante el cual un activo se asocia con un token digital registrado en *blockchain* de modo que el activo no se puede transferir sin transferir el token y viceversa. Por lo tanto, es un proceso que transforma la forma en que se registra la propiedad de un bien. Es comparable con la emisión de certificados en papel que representan valor ya que los certificados no son los valores en sí mismos, pero sirven como una prueba de propiedad de los valores. La tokenización de una activo produce que este activo sea más accesible a un público más amplio en un formato digital. [14]

La tokenización convierte el valor almacenado en un objeto tangible o intangible en un token, de manera que, es la conversión de cualquier activo en un token digital, y permite la transferencia, la propiedad y el almacenamiento digital sin la necesidad de un intermediario. El token es el producto de un *software* con una referencia única a bienes, propiedades, o derechos legales adjuntos[103].

6.4.3. Definición de *NFT*

Los *NFT* son tokens que se pueden utilizar para representar la propiedad de artículos únicos, permite tokenizar cosas como el arte, artículos de colección e incluso las bienes raíces. Solo se puede tener un propietario oficial a la vez y están protegidos por *blockchain*, de manera que nadie puede modificar el registro de propiedad. También, un *NFT* no es divisible. Un *NFT* puede representar tanto activos digitales como no digitales, algunos ejemplos son: arte digital, coleccionables, musica, vídeos, artículos del mundo real como escrituras de vehículos, entradas a eventos (como conciertos), facturas, documentos legales, firmas. Un *NFT* solo puede tener un propietario a la vez. La propiedad se gestiona a través de un ID único y los metadatos que ningún otro token puede replicar. Un *NFT* es publicado y gestionado por medio de un *smart contract* que asigna la propiedad y gestiona la transferabilidad [29].

Los tokens no fungibles son escrituras de propiedad programables basadas en *blockchain* para un

activo. Esta escritura digital otorga a su titular la capacidad exclusiva de usar, vender y transferir los derechos de propiedad del activo. Un *NFT* no contiene precisamente el activo que representa, sino que es un registro programable de propiedad con un puntero incorporado a la ubicación del activo [42].

6.4.4. Estándares de tokens

Los estándares de token más utilizados y de los que se derivan los demás, son los estándares de token realizados por Ethereum. Estos estándares son identificados por las siglas ERC por *Ethereum Request for Comment* seguido de un número. Los ERC son especificaciones para aplicaciones de Ethereum, como estándares de tokens, registros de nombres, formatos de bibliotecas, etc. El propósito de los ERC es que las aplicaciones y los *smart contracts* puedan comunicarse de forma predictiva, de tal forma que los ERC son documentos en donde se describen las reglas que se deben cumplir para implementar cierto token. Existen más de 10 ERC, pero son 2 los más utilizados para la creación de *NFTs*, el ERC-721 y el ERC-1155 que precisamente permiten crear tokens únicos que representan activos subyacentes. Si bien, son dos estándares principales para la creación de tokens no fungibles, es importante mencionar la existencia del ERC-20 ya que es el estándar más importante en la actualidad debido que todos los *smart contracts* se basan en este estándar. El ERC-20 es el estándar de los tokens fungibles del cual están creadas la mayoría de criptomonedas como el Ether y en contratos de este tipo solo se puede almacenar un tipo de activo. A diferencia del ERC-20, el ERC-721 tiene un número de identificador único que hace imposible que el token pueda ser fraccionado y sea único. Por otro lado, el ERC-1155 combina la capacidad tanto del ERC-20 como del ERC-721 de tal manera que el producto son tokens semi fungibles de tal manera que pueden existir varias ediciones de un solo *NFT*.

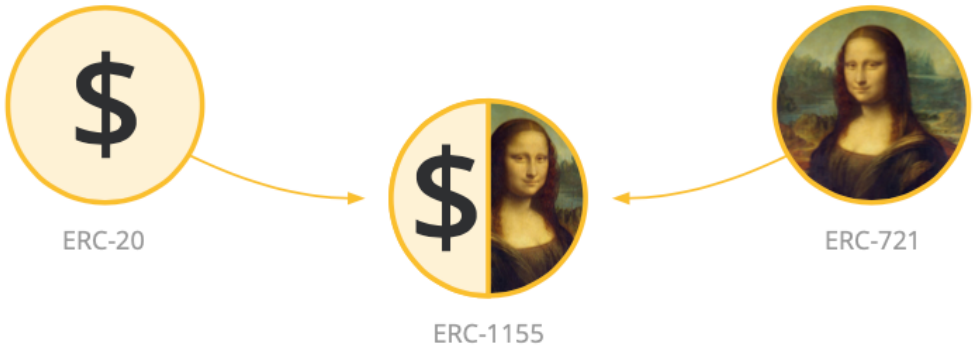


Figura 6.2: Estándares ERC de tokens [105]

6.4.5. Metadatos

Los metadatos son información que define y describe datos de tal manera que los metadatos de un *NFT* describen las propiedades o atributos del mismo, como nombre, imagen y descripción. Es decir, proporcionan información descriptiva para un *NFT* que permite a aplicaciones de comercio de *NFTs* obtener datos de los activos digitales y mostrarlos fácilmente en la aplicación. La metadata de un *NFT* puede ser almacenada de manera *on-chain* y *off-chain*, es decir, almacenar los datos

dentro de *blockchain* o de manera centralizada en algún servidor. El almacenamiento de los metadatos *on-chain* tiene como principal beneficio que los metadatos residen permanentemente junto con el token ya que los metadatos no pueden ser eliminados. Por otro lado, los metadatos *off-chain* es usualmente la forma de almacenamiento utilizada por la mayoría de proyectos ya que almacenar los metadatos en *blockchain* tiene sus limitaciones de espacio. Debido a que el almacenamiento fuera de la red de *blockchain* es el más utilizado, se han creado estándares de metadatos basados en el token, como el ERC-721. El estándar ERC-721 incluye un método denominado *tokenURI* que puede ser implementado para indicar el lugar en donde se pueden encontrar los metadatos de un elemento determinado. El método *tokenURI* retorna una URL pública que, a su vez, retorna un diccionario de datos JSON con los atributos del *NFT*.

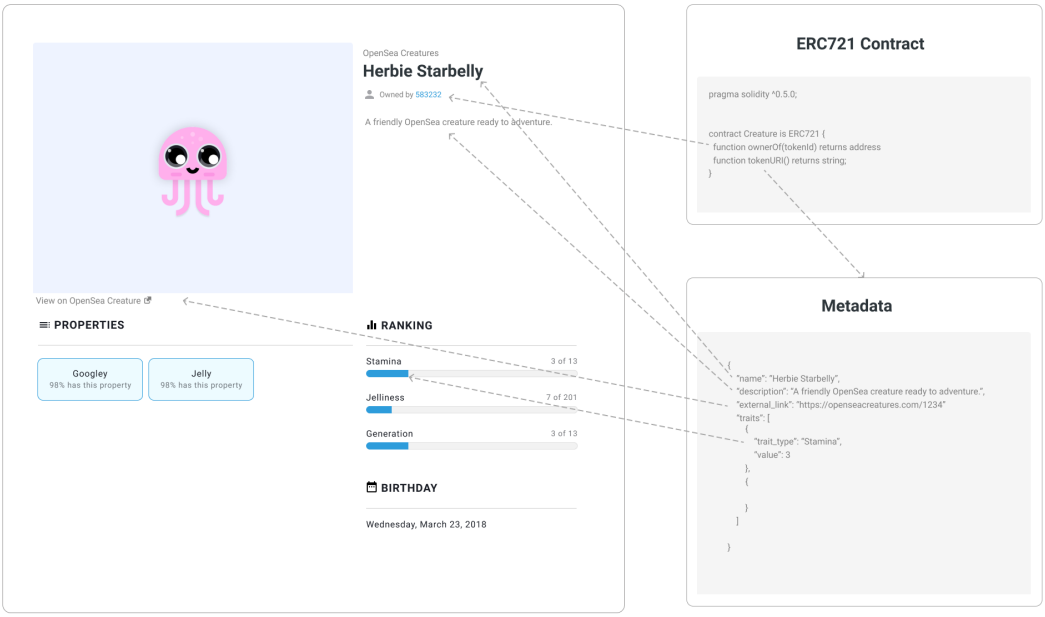


Figura 6.3: Almacenamiento de metadatos *off-chain* de un *NFT* [34]

6.5. *Web3*

El término "Web3" lo popularizó Gavin Wood poco tiempo después del lanzamiento de la red de Ethereum en el 2014, como solución para el problema que muchos de los primeros usuarios de las criptomonedas sintieron: el internet requería demasiada confianza. Es decir, la mayor parte del internet depende de confiar en unas empresas privadas para que actúen en el mejor interés del público. De esta manera, *Web3* es el término utilizado para la visión de un nuevo y mejorado internet. *Web3* utiliza *blockchain* para devolver el poder a los usuarios en forma de propiedad. *Web3* está descentralizado, de manera que, en lugar de el internet sea controlado y de propiedad de entidades centralizadas, la propiedad se distribuye entre los constructores y usuarios. *Web3* no tiene permisos de manera que todos tienen el mismo acceso para participar en *Web3* y nadie queda excluido. *Web3* no tiene confianza, opera utilizando incentivos y mecanismos económicos en lugar de depender de terceros de confianza [30].

6.6. Bases de datos

6.6.1. ¿Qué son bases de datos?

Se le llama base de datos a un conjunto de información recopilada que se almacena. Normalmente esta información recopilada pertenece a un mismo contexto y se almacena de forma ordenada para poder acceder a esta de forma sistemática durante recuperación, análisis, modificación o transmisión de datos. Actualmente, existen diversas formas de base de datos, desde una biblioteca hasta los datos de usuarios de una empresa [27].

Las bases de datos nacen de la necesidad humana de almacenar información y preservarla contra el tiempo y tener acceso fácil a esta. La computación brinda la capacidad necesaria para almacenar enormes cantidades de información en espacios físicos limitados.

La administración de estas enormes cantidades de datos digitales se lleva a cabo mediante sistemas de gestión: *Database Management Systems* (DBMS) o Sistemas de Gestión de Bases de Datos.

Al momento de diseñar una base de datos se deben tomar algunas decisiones debido a que las bases de datos pueden basarse en diferentes modelos y paradigmas, cada uno solucionando diferentes necesidades. Cada modelo de base de datos ofrece diferentes características, ventajas y limitaciones; principalmente en la estructura organizacional de los datos, jerarquías, capacidad de transmisión, capacidad de crecimiento, entre otros. Estos distintos modelos de base de datos se pueden clasificar en diferentes tipos.

6.6.2. Tipos de bases de datos

- Bases de datos relacionales
- Bases de datos orientadas a objetos
- Base de datos distribuidas
- Almacenes de datos
- Bases de datos NoSQL
- Bases de datos orientada a grafos
- Bases de datos OLTP
- Bases de datos de código abierto
- Bases de datos en la nube
- Base de datos multi-modelo
- Bases de datos de documentos/JSON
- Base de datos de autogestión [79]

6.7. *Server-Side Web Application Frameworks*

6.7.1. ¿Qué son los *Server-Side Web Application Frameworks*?

Un *Web Server* o Servidor *Web* es una computadora que contiene los archivos que componen un sitio *web*; esto incluye documentos HTML, imágenes, hojas de estilo y programas JavaScript. El servidor *web* se encarga de entregar todos estos archivos al dispositivo del usuario final para que pueda visualizar la información almacena remotamente. Esta información normalmente es accesible a través de un nombre de dominio que se mapea a una dirección IP de la máquina del servidor. El servidor también se encarga de controlar como los usuarios tienen acceso a los archivos y hace uso del protocolo HTTP (*Hypertext Transfer Protocol* o Protocolo de Transferencia de Hipertexto) [74].

Básicamente, cuando un usuario necesite información de un servidor *web*, el navegador realiza una petición HTTP al servidor y el servidor envía el archivo solicitado de regreso para el navegador pueda renderizarlo.

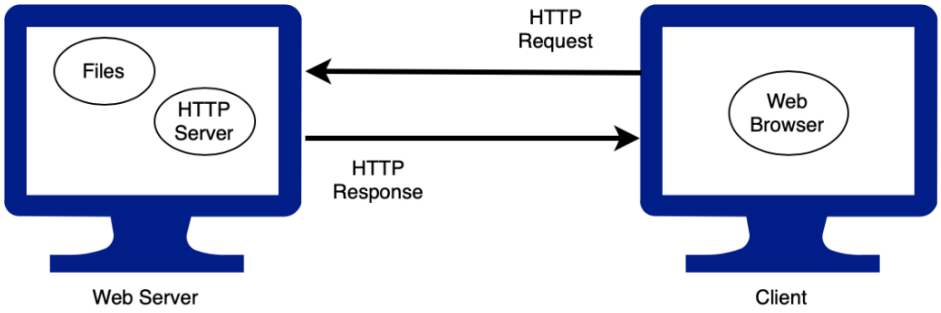


Figura 6.4: Comunicación cliente-servidor.

Las peticiones realizadas hacia un servidor *web* HTTP incluye principalmente la siguiente información:

URL identifica al servidor destino y un recurso dentro del servidor.

Método Define la acción a realizar sobre el recurso indicado en la URL. Este método puede ser *GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH*; y cada uno tiene su propósito, por ejemplo, obtener, guardar, eliminar o actualizar datos. [72]

Server-Side Web Application Frameworks o *Frameworks* de Aplicaciones *Web* de lado del servidor son *frameworks software* que facilitan escribir, mantener y escalar aplicaciones *web*. Estos *frameworks* proporcionan herramientas y librerías que simplifican tareas comunes de programación *web*; entre estas se pueden mencionar enrutado de URLs a los manejadores correspondientes, interacción con base de datos, sesiones y autorizaciones de usuario, formateado de respuestas HTML, JSON, XML y seguridad contra ataques *web* [76].

6.7.2. API

Las APIs o Interfaz de Programación de Aplicaciones, son interfaces expuestas que permiten abstraer funcionalidad de sistemas complejos. Las APIs proveen una sintaxis fácil de usar, que simplifican el proceso de comunicación entre dos sistemas diferentes e independientes. Las APIs constan de definiciones y protocolos que se pueden considerar como contratos con documentación que representan un acuerdo entre dos partes, e indican cómo se comunicarán [91].

La arquitectura de API RESTful debe cumplir las siguientes condiciones:

- Se monta sobre una arquitectura cliente-servidor y administra las peticiones con HTTP.
- Las peticiones son independientes entre sí, no se almacena estado ni se comparte entre las sesiones levantadas por peticiones distintas.
- Se monta sobre una arquitectura que tiene la capacidad de almacenamiento cache, evitando algunas interacciones cliente-servidor.
- Sistema en capas.
- Disponibilidad de código.
- Interfaz uniforme [91].

6.8. *Cloud Computing*

6.8.1. ¿Qué es *Cloud Computing*?

Para la publicación de páginas o aplicaciones *web* se necesitan sistemas de cómputo funcionando todo el tiempo que satisfacen las necesidades de las peticiones de clientes. Tradicionalmente, se compraban servidores físicos y se montaba una infraestructura de red física. Esto requiere espacio, mantenimiento y recursos. *Cloud Computing* ofrece una solución digital a este problema.

Cloud Computing es la disponibilidad bajo demanda de recursos de computación como servicios a través de internet. Esta tecnología evita que las empresas tengan que encargarse de aprovisionar, configurar o gestionar los recursos y permite que paguen únicamente por los que usen [18].

Hay tres tipos de modelos de servicio de *Cloud Computing*:

- Infraestructura como servicio: ofrece servicios de computación y almacenamiento.
- Plataforma como servicio: proporciona un entorno de desarrollo y despliegue para crear aplicaciones en la nube.
- *Software* como servicio: facilita aplicaciones como servicios.

Entre las ventajas que ofrece *Cloud Computing* se pueden mencionar:

- Flexibilidad
- Eficacia

- Valor estratégico
- Seguridad
- Rentabilidad

Usos del *Cloud Computing*:

- Escalado de infraestructuras
- Recuperación tras fallos
- Almacenamiento de datos
- Desarrollo de aplicaciones
- Análisis de *Big Data*

6.9. Inteligencia artificial

Coloquialmente, el término inteligencia artificial se aplica cuando una máquina o programa informático imita las funciones cognitivas que los humanos asocian con otras mentes humanas, como por ejemplo: percibir, razonar, aprender y resolver problemas. Andreas Kaplan y Michael Haenlein definen la inteligencia artificial como *la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible.* [87]

6.9.1. Aprendizaje supervisado

El aprendizaje supervisado, también conocido como aprendizaje automático supervisado, es una subcategoría del aprendizaje automático y la inteligencia artificial. Se define por su uso de conjuntos de datos etiquetados para entrenar algoritmos que clasifican datos o predicen resultados con precisión. A medida que los datos de entrada se introducen en el modelo, éste ajusta sus parámetros hasta que el modelo se haya ajustado correctamente, lo que ocurre como parte del proceso de validación cruzada. El aprendizaje supervisado ayuda a las organizaciones a resolver una variedad de problemas del mundo real a escala, como clasificar el correo no deseado en una carpeta separada de su bandeja de entrada [25].

6.9.2. Aprendizaje no supervisado

El aprendizaje no supervisado trabaja con datos que no han sido etiquetados. Estos algoritmos son usados principalmente en tareas donde es necesario analizar los datos para extraer nuevo conocimiento o agrupar entidades por afinidad.

También tiene aplicaciones para reducir la dimensionalidad de conjuntos de datos o simplificar conjuntos de datos. En el caso de agrupar datos por afinidad, el algoritmo debe definir una métrica de similitud o distancia que le sirva para comparar los datos entre sí. Como ejemplo de aprendizaje no supervisado se tienen los algoritmos de agrupamiento o *clustering*, que podrían aplicarse

para encontrar clientes con características similares a los que ofrecer determinados productos o destinar una campaña de *marketing*, descubrimiento de tópicos o detección de anomalías, entre otros [98].

6.9.3. Aprendizaje semi supervisado

Este tipo de aprendizaje tiene un poco de los dos anteriores. Usando este enfoque, se comienza etiquetando manualmente una porción de los datos. Posteriormente, se entrena uno o varios algoritmos de aprendizaje supervisado sobre esa pequeña parte de datos etiquetados y se usan los modelos resultantes del entrenamiento para etiquetar el resto de datos. Finalmente, se entrena un algoritmo de aprendizaje supervisado utilizando como etiquetas las etiquetadas manualmente, más las generadas por los modelos anteriores [98].

6.9.4. Aprendizaje reforzado

El aprendizaje por refuerzo es un método de aprendizaje automático que se basa en recompensar los comportamientos deseados y penalizar los no deseados. Aplicando este método, un agente es capaz de percibir e interpretar el entorno, ejecutar acciones y aprender a través de prueba y error. Es un aprendizaje que fija objetivos a largo plazo para obtener una recompensa general máxima y lograr una solución óptima. Sus aplicaciones más comunes pueden ser encontradas en juegos o simulaciones. En estos casos, el agente recibe información sobre las reglas del juego y aprende a jugar por sí mismo. Al principio se comporta de manera aleatoria, pero con el tiempo aprenderá movimientos más sofisticados. Este tipo de aprendizaje se aplica también en otras áreas como la robótica, la optimización de recursos o sistemas de control [98].

6.10. Red neuronal

Una red neuronal es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de una manera que está inspirada en la forma en que lo hace el cerebro humano. Se trata de un tipo de proceso de *machine learning* llamado aprendizaje profundo, que utiliza los nodos o las neuronas interconectados en una estructura de capas que se parece al cerebro humano. Crea un sistema adaptable que las computadoras utilizan para aprender de sus errores y mejorar continuamente. De esta forma, las redes neuronales artificiales intentan resolver problemas complicados, como la realización de resúmenes de documentos o el reconocimiento de rostros, con mayor precisión. [9].

El cerebro humano es lo que inspira la arquitectura de las redes neuronales. Las células del cerebro humano, llamadas neuronas, forman una red compleja y con un alto nivel de interconexión y se envían señales eléctricas entre sí para ayudar a los humanos a procesar la información. De manera similar, una red neuronal artificial está formada por neuronas artificiales que trabajan juntas para resolver un problema. Las neuronas artificiales son módulos de *software*, llamados nodos, y las redes neuronales artificiales son programas de *software* o algoritmos que, en esencia, utilizan sistemas informáticos para resolver cálculos matemáticos.

Las redes neuronales profundas, o redes de aprendizaje profundo, tienen varias capas ocultas con millones de neuronas artificiales conectadas entre sí. Un número, denominado peso, representa las

conexiones entre un nodo y otro. El peso es un número positivo si un nodo estimula a otro, o negativo si un nodo suprime a otro. Los nodos con valores de peso más altos tienen mayor influencia en los demás nodos. En teoría, las redes neuronales profundas pueden asignar cualquier tipo de entrada a cualquier tipo de salida. Sin embargo, también necesitan mucho más entrenamiento en comparación con otros métodos de *machine learning*. Necesitan millones de ejemplos de datos de entrenamiento en lugar de los cientos o miles que podría necesitar una red más simple [9].

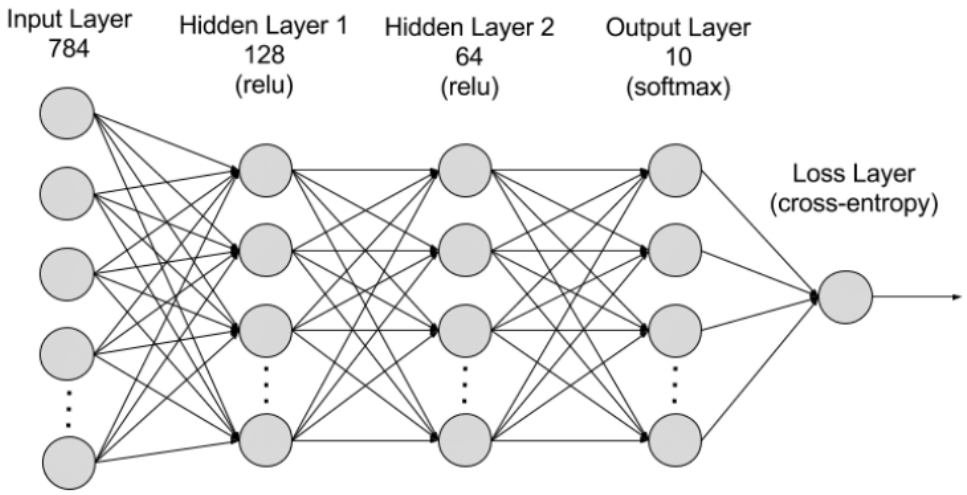


Figura 6.5: Ejemplo de una red neuronal profunda [9]

6.10.1. Función de activación

En un nodo, la función de activación se encarga de tomar todos los valores de entrada (*inputs*), y producir un valor de salida (*output*). Entre las funciones de activación populares se encuentran:

- Gaussiana
$$f(x) = e^{-x^2}$$
- Multicuadrática
$$f(x) = \sqrt{1 + (\epsilon x)^2}$$
- Multicuadrática inversa
$$f(x) = \frac{1}{\sqrt{1 + (\epsilon x)^2}}$$
- Tangente hiperbólica
$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$
- Sigmoide
$$f(x) = \frac{1}{1 + e^{-}x}$$
- Función rampa
$$f(x) = max(0, x)$$

- Función signo

$$f(x) = sgn(x) = \begin{cases} 1 & si \quad x > 0 \\ 0 & si \quad x = 0 \\ -1 & si \quad x < 0 \end{cases}$$

[41]

6.10.2. Perceptrón simple

El perceptrón simple consta de una red con una capa de entrada de **n** neuronas y otra con **m** neuronas de salida. Usa señales binarias tanto de entrada como de salida, y su función de activación es usualmente de tipo signo. Fue revelado inútil cuando se recogieron sus limitaciones:

El perceptrón simple sólo sirve para clasificar problemas linealmente separables, cosa que ya se podía hacer mediante métodos estadísticos, y de una forma mucho más eficiente [68].

6.10.3. Perceptrón multicapa

Es la arquitectura más simple que se encuentra de red neuronal en la actualidad. En ésta se cuenta, por convención, con un una lista de neuronas inicial (llamada *capa de entrada*).

Posteriormente se encuentra una o más capas en las que, en cada una, cada neurona se encuentra conectada con todas las neuronas de la capa anterior. Es decir, que cada neurona recibe como valores de entrada, todos los valores de salida de la capa que le precede. Éstas son llamadas *capas ocultas*.

Finalmente se encuentra una *capa de salida*, conformada por una o más neuronas cuyo valor de salida es de valor directo para el problema que se desee resolver [16].

6.10.4. Red neuronal convolucional

Las redes neuronales convolucionales son un tipo de redes neuronales donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación puede ser realizada en matrices bidimensionales, han demostrado ser muy efectivas en tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones [12].

Consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa por lo general se añade una función para realizar un mapeo causal no-lineal. Al principio estas redes tienen una fase de extracción de características, compuesta de neuronas convolucionales, luego hay una reducción por muestreo y al final tendremos neuronas de perceptrón más sencillas para realizar la clasificación final sobre las características extraídas.

La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

6.10.5. Retro propagación (propagación hacia atrás)

Es un método de cálculo de descenso de gradiente usado en algoritmos de aprendizaje supervisado que buscan entrenar redes neuronales. El método emplea un ciclo propagación/adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de las capas ocultas que contribuyen a la salida. Sin embargo las neuronas de las capas ocultas sólo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total [32].

6.11. Algoritmos de regresión

En *Machine Learning* se usan varios tipos de algoritmos para permitir que las máquinas aprendan relaciones y hagan predicciones basadas en patrones o reglas identificadas a partir del conjunto de datos. La regresión es una técnica de aprendizaje automático en la que el modelo predice la salida como un valor numérico continuo.

El análisis de regresión se usa a menudo en finanzas, inversiones y otros, y descubre la relación entre una sola variable dependiente (variable objetivo) que depende de varias independientes. Por ejemplo, predecir el precio de la vivienda, el mercado de valores o el salario de un empleado, etc. son los más comunes problemas de regresión [99].

6.11.1. Regresión lineal

El análisis de regresión lineal se utiliza para predecir el valor de una variable en función del valor de otra variable. La variable que desea predecir se llama variable dependiente. La variable que estás usando para predecir el valor de la otra variable se llama variable independiente.

Esta forma de análisis estima los coeficientes de la ecuación lineal, involucrando una o más variables independientes que predicen mejor el valor de la variable dependiente. La regresión lineal se ajusta a una línea recta o superficie que minimiza las discrepancias entre los valores de salida previstos y reales. Existen calculadoras de regresión lineal simples que utilizan un método de "mínimos cuadrados"para descubrir la línea que mejor se ajusta a un conjunto de datos emparejados. Luego

estima el valor de X (variable dependiente) a partir de Y (variable independiente) [51].

6.11.2. Árbol de decisión

Un árbol de decisión es un algoritmo de aprendizaje supervisado no paramétrico, que se utiliza tanto para tareas de clasificación como de regresión. Tiene una estructura de árbol jerárquica, que consta de un nodo raíz, ramas, nodos internos y nodos hoja.

Un árbol de decisión comienza con un nodo raíz, que no tiene ramas entrantes. Las ramas salientes del nodo raíz alimentan los nodos internos, también conocidos como nodos de decisión. En función de las características disponibles, ambos tipos de nodos realizan evaluaciones para formar subconjuntos homogéneos, que se indican mediante nodos hoja o nodos terminales. Los nodos hoja representan todos los resultados posibles dentro del conjunto de datos.

El aprendizaje del árbol de decisiones emplea una estrategia de divide y vencerás mediante la realización de una búsqueda codiciosa para identificar los puntos de división óptimos dentro de un árbol. Este proceso de división se repite de forma recursiva de arriba hacia abajo hasta que todos o la mayoría de los registros se hayan clasificado bajo etiquetas de clase específicas. Que todos los puntos de datos se clasifiquen o no como conjuntos homogéneos depende en gran medida de la complejidad del árbol de decisión. Los árboles más pequeños son más fáciles de obtener nodos hoja puros, es decir, puntos de datos en una sola clase. Sin embargo, a medida que un árbol crece en tamaño, se vuelve cada vez más difícil mantener esta pureza y, por lo general, da como resultado que haya muy pocos datos dentro de un subárbol determinado. Cuando esto ocurre, se conoce como fragmentación de datos y, a menudo, puede resultar en sobreajustes (*overfitting*) [53].

6.11.3. Métodos de aprendizaje en conjunto

El aprendizaje conjunto se basa en la idea de la "*sabiduría de las multitudes*", lo que sugiere que la toma de decisiones de un grupo más grande de personas suele ser mejor que la de un experto individual. De manera similar, el aprendizaje conjunto se refiere a un grupo (o conjunto) de aprendices básicos, o modelos, que trabajan colectivamente para lograr una mejor predicción final. Es posible que un solo modelo, también conocido como modelo básico o de aprendizaje débil, no funcione bien individualmente debido a la alta varianza o al alto sesgo. Sin embargo, cuando se agregan los alumnos débiles, pueden formar un alumno fuerte, ya que su combinación reduce el sesgo o la varianza, lo que produce un mejor rendimiento del modelo.

Los métodos de conjunto se ilustran con frecuencia utilizando árboles de decisión, ya que este algoritmo puede ser propenso a sobre ajustarse (alta varianza y bajo sesgo) cuando no se ha podado y también puede prestarse a no ajustarse (baja varianza y alto sesgo) cuando es muy pequeño. Cuando un algoritmo se sobreajusta o no se ajusta a su conjunto de entrenamiento, no puede generalizarse bien a nuevos conjuntos de datos, por lo que se utilizan métodos de conjunto para contrarrestar este comportamiento y permitir la generalización del modelo a nuevos conjuntos de datos. Si bien los árboles de decisión pueden exhibir una gran varianza o un alto sesgo, vale la pena señalar que no es la única técnica de modelado que aprovecha el aprendizaje conjunto para encontrar el "punto óptimo"dentro de la compensación entre sesgo y varianza [49].

Embolsado

Más comúnmente conocido como embolsado por su nombre en inglés (*bagging* como acrónimo de *Bootstrap Aggregation*), es un método de aprendizaje conjunto usado para reducir varianza en un conjunto de datos con mucho ruido. Funciona tomando una muestra aleatoria del conjunto de datos original con reemplazo (de manera que puntos individuales pueden ser escogidos repetidas veces) y entrenando un modelo por cada muestra. Después de generar varias muestras, estos modelos débiles se entrenan de forma independiente y, según el tipo de tarea (regresión o clasificación, por ejemplo), el promedio o la mayoría de esas predicciones arrojan una estimación más precisa que la que arrojaría un solo modelo entrenado con el *dataset* completo. El algoritmo de *random forest* se considera una extensión de este método, puesto que utiliza tanto el embolsado como la aleatoriedad de características para crear un bosque de árboles de decisión no correlacionados [50].

Random Forest

Random forest es un algoritmo de aprendizaje automático de uso común que combina la salida de múltiples árboles de decisión para llegar a un único resultado. Su facilidad de uso y flexibilidad han impulsado su adopción, ya que maneja problemas de clasificación y regresión.

El algoritmo de *Random Forest* es una extensión del método de embolsado (*bagigng*), ya que utiliza tanto el embolsado como la aleatoriedad de características para crear un bosque de árboles de decisión no correlacionados. La aleatoriedad de características, también conocida como empaquetado de características o "el método del subespacio aleatorio", genera un subconjunto aleatorio de características, lo que garantiza una baja correlación entre los árboles de decisión. Esta es una diferencia clave entre los árboles de decisión y los bosques aleatorios.

Los algoritmos de *random forest* tienen, principalmente, tres hiperparámetros que deben configurarse antes del entrenamiento. Estos incluyen el tamaño del nodo, la cantidad de árboles y la cantidad de características muestreadas. A partir de ahí, el modelo puede utilizarse para resolver problemas de regresión o clasificación.

El algoritmo se compone de una colección de árboles de decisión, y cada árbol del conjunto se compone de una muestra de datos extraída de un conjunto de entrenamiento con reemplazo, denominada muestra de arranque. De esa muestra de entrenamiento, cerca de tercio se reserva como datos de prueba, conocidos como la muestra fuera de la bolsa (*oob* por sus siglas en inglés*Out Of the Box*), a la que se regresará posteriormente. Luego, se inyecta otra instancia de aleatoriedad a través del embolsado de características, lo que agrega más diversidad al conjunto de datos y reduce la correlación entre los árboles de decisión. Dependiendo del tipo de problema, la determinación de la predicción variará. Para una tarea de regresión, se promediarán los árboles de decisión individuales, y para una tarea de clasificación, un voto mayoritario, es decir, la variable categórica más frecuente, producirá la clase predicha. Finalmente, la muestra *oob* se usa para la validación cruzada, finalizando esa predicción[52].

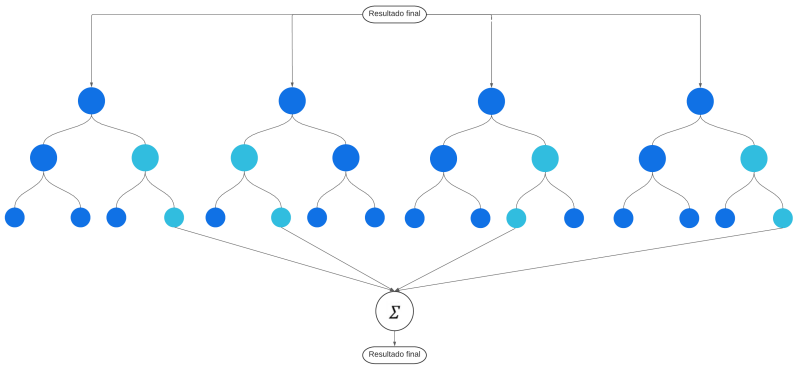


Figura 6.6: Algoritmo de *random forest*.

XGBoost

XGBoost es un enfoque poderoso para construir modelos de regresión supervisados. La validez de esta declaración se puede inferir al conocer su función objetivo (*XGBoost*) y los alumnos base. La función objetivo contiene una función de pérdida y un término de regularización. Informa sobre la diferencia entre los valores reales y los valores predichos, es decir, qué tan lejos están los resultados del modelo de los valores reales. La función de pérdida más común en *XGBoost* para problemas de regresión es *reg:linear*, y la de clasificación binaria es *reg:logistics*. El aprendizaje de conjunto implica entrenar y combinar modelos individuales (conocidos como estudiantes base) para obtener una única predicción, y *XGBoost* es uno de los métodos de aprendizaje de conjunto. Este algoritmo espera tener los alumnos base que son uniformemente malos, de modo que cuando se combinan todas las predicciones, las malas predicciones se cancelan y las mejores se suman para formar buenas predicciones finales [36].

6.12. Controles de acceso

Los controles de acceso son funciones de seguridad que controlan cómo los usuarios y los sistemas se comunican e interactúan con otros sistemas y recursos. Protegen los sistemas y recursos del acceso no autorizado y pueden ser componentes que participen en la determinación del nivel de autorización después de que se haya completado con éxito un procedimiento de autenticación. Los controles de acceso brindan a las organizaciones la capacidad de controlar, restringir, monitorear y proteger la disponibilidad, integridad y confidencialidad de los recursos [39].

6.12.1. Proceso

Para que un usuario pueda acceder a un recurso, primero debe demostrar que es quien dice ser, tiene las credenciales necesarias y se le han otorgado los derechos o privilegios necesarios para realizar las acciones que solicita. La identificación describe un método para garantizar que un sujeto es la entidad que dice ser. La identificación se puede proporcionar con el uso de un nombre de usuario o número de cuenta. Para estar correctamente autenticado, generalmente se requiere que el sujeto proporcione una segunda parte del conjunto de credenciales. Esta llave podría ser una contraseña, una frase de contraseña, una clave criptográfica, un número de identificación personal (PIN), un

atributo anatómico o un token. Una vez que el sujeto proporciona sus credenciales y está debidamente identificado, el sistema al que intenta acceder debe determinar si se le han otorgado a este sujeto los derechos y privilegios necesarios para llevar a cabo las acciones solicitadas. Si el sistema determina que el sujeto puede acceder al recurso, autoriza al sujeto [39].

6.12.2. KYC

KYC significa "Know Your Client.º Çonozca a su cliente". Describe el proceso de verificación de la identidad de nuevos clientes. El proceso KYC se realiza para prevenir actividades ilegales como el lavado de dinero o el fraude, protegiendo a cambio tanto a la empresa como al cliente, por lo que es importa en organizaciones y plataformas que tratan con dinero. Esta verificación puede ser a través de algún documento de identificación del país, reconocimientos facial o por una dirección [54].

6.13. Criptografía

La criptografía es un método de almacenamiento y transmisión de datos en una forma que solo aquellos para los que está destinado pueden leer y procesar. Se considera una ciencia de proteger la información al codificarla en un formato ilegible. La criptografía es una forma eficaz de proteger la información confidencial, ya que se almacena en medios o se transmite a través de rutas de comunicación de red no confiables. El objetivo de la criptografía es hacer que la obtención de la información requiera mucho trabajo o demasiado tiempo para cualquier atacante. Consiste en llevar un texto leíble, que se le conoce como texto plano, el cual cualquier humano y computadora puede entender, y convertirlo a una forma ilegible llamada texto cifrado, la cual ni un humano o máquina puede entender [39].

6.13.1. Principio de Kerckhoffs

Este principio formulado por Auguste Kerckhoffs nos dice que un sistema criptográfico debe ser seguro incluso si todo lo relacionado con el sistema, excepto la clave, es de conocimiento público. Dice que solo debe haber un secreto que es la clave, ya que mientras más secretos haya más propenso es el algoritmo a poder ser comprometido [39].

6.13.2. Tipos de cifrado

Un cifrado de sustitución utiliza una clave para dictar cómo debe llevarse a cabo la sustitución. La sustitución se usa en los algoritmos simétricos de hoy. En un cifrado de transposición, los valores se codifican o se colocan en un orden diferente. La clave determina las posiciones a las que se mueven los valores. Cuando se implementan con funciones matemáticas complejas, las transposiciones pueden volverse bastante sofisticadas y difíciles de romper. Los algoritmos simétricos empleados en la actualidad utilizan secuencias largas de sustituciones y transposiciones complicadas en los mensajes [39].

6.13.3. Métodos de cifrado

En un criptosistema que usa criptografía simétrica, el remitente y el receptor usan dos instancias de la misma clave para el cifrado y descifrado. Por lo tanto, la clave tiene una doble funcionalidad, ya que puede realizar procesos de cifrado y descifrado. Las claves simétricas también se denominan claves secretas, porque este tipo de cifrado depende de que cada usuario mantenga la clave en secreto y debidamente protegida. Si un intruso obtuviera esta clave, podría descifrar cualquier mensaje interceptado cifrado con ella. En la criptografía de clave simétrica, se utiliza una única clave secreta entre entidades, mientras que en los sistemas de clave pública, cada entidad tiene claves diferentes o claves asimétricas. Las dos claves asimétricas diferentes están relacionadas matemáticamente. Si un mensaje está cifrado con una clave, se requiere la otra clave para descifrar el mensaje. En un sistema de clave pública, el par de claves se compone de una clave pública y una clave privada. La clave pública puede ser conocida por todos, y la clave privada debe ser conocida y utilizada solo por el propietario [39].

6.14. Análisis de riesgos

La gestión del riesgo de la información es el proceso de identificar y evaluar el riesgo, reducirlo a un nivel aceptable e implementar los mecanismos correctos para mantener ese nivel, ya que no existe un entorno 100 por ciento seguro. Cada entorno tiene vulnerabilidades y amenazas hasta cierto punto. La habilidad consiste en identificar estas amenazas, evaluar la probabilidad de que realmente ocurran y el daño que podrían causar, y luego tomar las medidas adecuadas para reducir el nivel general de riesgo en el entorno a lo que la organización identifica como aceptable. Los riesgos para una empresa se presentan de diferentes formas y no todos están relacionados con la informática. El análisis de riesgos, es una herramienta para la gestión de riesgos, es un método para identificar vulnerabilidades y amenazas y evaluar los posibles impactos para determinar dónde implementar salvaguardas de seguridad. El análisis de riesgos se utiliza para garantizar que la seguridad sea rentable, relevante, oportuna y receptiva a las amenazas. El análisis de riesgos ayuda a las empresas a priorizar sus riesgos y muestra la cantidad de dinero que se debe aplicar para protegerse contra esos riesgos de manera sensata [39].

6.15. Políticas de seguridad

Una política de seguridad es un documento que detalla las reglas, las expectativas y el enfoque general que utiliza una organización para mantener la confidencialidad, integridad y disponibilidad de sus datos. Las políticas de seguridad existen en muchos niveles diferentes, desde construcciones de alto nivel que describen los objetivos y principios generales de seguridad de una empresa hasta documentos que abordan problemas específicos, como el uso del internet [45].

6.16. SQL *Injection*

Un ataque de inyección SQL consiste en la inserción de una consulta SQL a través de los datos de entrada del cliente a la aplicación. Un *exploit* de inyección SQL exitoso puede leer datos confidenciales de la base de datos, modificar datos de la base de datos, es decir insertar, actualizar o eliminar datos, ejecutar operaciones de administración en la base de datos, o recuperar el contenido de un

archivo dado presente en el archivo DBMS sistema. Los ataques de inyección SQL son un tipo de ataque de inyección, en el que los comandos SQL se inyectan en los ingresos de datos por el usuario para afectar la ejecución de los comandos SQL predefinidos [80].

6.17. Triada CIA

Esta triada representa la confidencialidad, integridad y disponibilidad (*availability*). Juntos, estos tres principios son fundamentales para la infraestructura de seguridad de cualquier organización. Deberían funcionar como metas y objetivos para cada programa de seguridad. Cuando ocurre algún ataque seguramente una o más de estos principios fueron violados. Se deben evaluar las amenazas y las vulnerabilidades en función del impacto potencial que tienen sobre estos tres principios y como pueden afectar a los activos de la organización [102].

6.17.1. Confidencialidad

La confidencialidad se refiere a mantener los datos privados o secretos. Se trata de controlar el acceso a los datos para evitar la divulgación no autorizada. Esto implica garantizar que solo aquellos que están autorizados tengan acceso a activos específicos y que aquellos que no están autorizados no puedan obtener acceso activamente. La confidencialidad puede ser violada por medio de diferentes ataques para poder obtener acceso a la información. Pero la confidencialidad también se puede violar involuntariamente por error humano, descuido o controles de seguridad inadecuados. Para mejorar la confidencialidad, se pueden usar emplear accesos de control y mecanismos de autorización seguros, así como el cifrado de la información [102].

6.17.2. Integridad

La integridad se refiere a la cualidad de que algo sea íntegro o completo. En la seguridad informática, la integridad se trata de garantizar que los datos no hayan sido manipulados y, por lo tanto, se pueda confiar en ellos. Que sean correctos, auténticos y confiables. Garantizar la integridad implica proteger los datos en uso, en tránsito y cuando se almacenan. La integridad se puede comprometer a través de ataques o por error humano. Algunas de las contramedidas que protegen la integridad de los datos incluyen cifrado, *hashing*, firmas digitales y certificados digitales [102].

6.17.3. Disponibilidad

Disponibilidad significa que las redes, los sistemas y las aplicaciones están en funcionamiento. Garantiza que los usuarios autorizados tengan acceso oportuno y confiable a los recursos cuando se necesitan. Muchas cosas pueden poner en peligro la disponibilidad, incluidas fallas de *hardware* o *software*, fallas de energía, ataques, desastres naturales y errores humanos. Entre las contramedidas para ayudar a garantizar la disponibilidad incluyen redundancia, tolerancia a fallas de *hardware*, parches de *software* regulares y actualizaciones del sistema, copias de seguridad, planes de recuperación ante desastres [102].

CAPÍTULO 7

Metodología

7.1. Administración de proyecto

7.1.1. Tecnologías

En la actualidad, existe una gran cantidad de diferentes tecnologías que nos ayudan a tener un ambiente cómodo de desarrollo. El propósito de esta sección en el proyecto es el de dar guía a aquellos que están iniciando en el desarrollo de *software* o quisieran tener alternativa a sus tecnologías y procesos actuales, iniciando a utilizar otras tecnologías, herramientas, estándares y procesos.

Los puntos que más se destacan en cuanto se refiere este tema son el editor de código fuente y el servicio de control de versiones que utilizamos, entre otras herramientas que se mencionan posteriormente. Es importante la elección de estas herramientas ya que una mala elección nos pondrá más trabajo a nosotros, y una buena elección nos facilitarán procesos repetitivos y tediosos. Además es importante hacer la selección de estas herramientas como un todo ya que existen herramientas y tecnologías que se integran mejor entre sí.

A continuación se presentan las tecnologías, herramientas, estándares y procesos usados durante la elaboración de este proyecto y sus alternativas. Es importante mencionar que un ambiente de desarrollo puede ser personal o de equipo y las decisiones que se toman son subjetivas, por gustos y preferencias de la persona o del equipo.

Estándares de desarrollo

Entre los estándares de desarrollo que se contemplaron para la elaboración de este proyecto se puede mencionar la utilización de control de versiones basado en git. Git es un *software* para control de versiones distribuido. Esta herramienta nos ayuda a llevar un historial de los cambios realizados

en los archivos y distribuir el código fuente y sus actualizaciones fácilmente con todo el equipo de trabajo. El uso en si de git es un valor añadido a nuestro código ya que se pueden crear versiones y mantenerlas, agilizando el flujo de desarrollo.

Sobre git se puede definir el estándar denominado –*Conventional Commits*–, este estándar define el uso principal de git, que son los *commits*. *Conventional Commits* define la estructura que debe llevar cada *commit* en el momento de realizarlo, principalmente el mensaje que este guarda con la información más relevante de lo que ocurrió en ese *commit*. Este estándar es definido por el equipo.

Otro estándar muy importante se trata sobre el código en si, su estructura y como se ve, por ejemplo como se definen los nombres de variables y funciones, el uso de comentarios y la documentación que se agrega al código.

Editores de código fuente

El editor de código es una elección personal mayormente. Existen muchos editores de código y todos cumplen su función principal muy bien. Sin embargo, hay unos editores que proveen funcionalidades que nos facilitan el trabajo, como mayor integración con otras herramientas utilizadas en el ciclo de vida de desarrollo. A continuación se listan algunas opciones de editores de código fuente que proveen funcionalidades muy completas y varias integraciones útiles.

Sublime Text [107] es un editor de código simple, con pocas opciones abrumadoras, muy bueno para personas que están iniciando en el desarrollo de *software*.

Atom [7] es un editor que presenta más opciones que *Sublime Text*, con una muy buena integración con git y Github, sin embargo no presenta tantas opciones como otros editores.

Visual Studio Code [19] conocido como VS Code, es un editor de alta calidad, con integraciones muy útiles y *marketplace* de extensiones que se pueden agregar para incorporar mayor funcionalidad gradualmente o a necesidad del programador.

JetBrains [57] es una empresa que provee editores de código dedicados, para condiciones muy específicas a partir del lenguaje de programación a utilizar. Tiene editores muy potentes sin embargo pueden llegar a ser abrumadores para desarrolladores que están iniciando en el mundo de la programación.

Servicios en nube para control de versiones

Existe algunas opciones de servicio de almacenamiento en nube con control de versiones. Estos son muy útiles y su uso es altamente recomendado, ya que nos quita responsabilidad sobre la seguridad del código. Estos servicios almacenan una copia de nuestro código que podemos descargar y administrar en nube, asimismo nos facilita el proceso de distribución de código y actualización cuando se trabaja en equipo.

Es importante saber que hay que hacer uso adecuado de estos servicios ya que también pueden dejar de ser beneficiosos si no se usan adecuadamente. Entre los puntos más importantes a mencionar en estos aspectos son la configuración de visibilidad del código, dígase tener un repositorio privado o público; no subir archivos sensibles como credenciales y una adecuada configuración de ramas como requerir revisión para poder actualizar el código base y otros. A continuación se listan algunos

servicios de almacenamiento en nube con control de versiones para código fuente. Los servicios presentados son muy completos y proveen diferentes herramientas e integraciones útiles.

Github [37] presenta múltiples herramientas integradas que facilitan el desarrollo por su fácil acceso, uso e integración.

GitLab [38] presenta herramientas muy completas para flujos de integración continua y *deployment* continuo.

Bitbucket [5] presenta planes gratuitos más completos comparado a las otras herramientas, además provee un ambiente de desarrollo completo con las diferentes herramientas de Atlassian.

Herramientas para seguimiento de requerimientos

Un punto tan crítico como es la planificación y el seguimiento de requerimientos en el flujo de desarrollo requiere de atención primordial. La selección de esta herramienta es importante ya que facilitará el seguimiento de los requerimientos y la motivación del equipo. Hay variedad de formas en las que se puede hacer seguimiento de requerimientos, desde mover *post-its* en un pizarrón a herramientas que se integren con el ambiente de desarrollo y controlador de versiones. A continuación se listan algunas opciones.

Trello [109] es una herramienta que nos puede servir para seguimiento de tareas, es simple y se puede integrar con otras.

Jira [6] es una herramienta muy completa, dedicada al ambiente de desarrollo con Atlassian. Puede ser un poco abrumadora por la cantidad de opciones que presenta.

Monday [69] es una herramienta muy completa, y se puede integrar con otros sistemas ya que por si solo, al igual que Trello no proveen el ambiente completo de desarrollo.

Github Projects [37] es una herramienta muy completa, dedicada al ambiente de desarrollo con Github. Provee múltiples herramientas fáciles de usar que se integran muy bien con el desarrollo diario.

Adicionalmente, hay algunas metodologías de trabajo ágil que se pueden poner en práctica con estas herramientas, como son *SCRUM* y *KANBAN*. La metodología a utilizar debería ser tomada en cuenta al momento de elección de la herramienta de seguimiento de requerimientos.

Visualizadores de base de datos

La visualización de base de datos hace más fácil el trabajo del desarrollador, ya que con un buen visualizador de base de datos se podrá ver con facilidad el diseño de la base de datos, las relaciones, y los registros. Además, se puede *debuggear* con mayor facilidad.

Cada DBMS, o Sistema Administrador de Base de Datos de sus siglas en inglés, provee un visualizador de base de datos. Así que puede depender del DBMS elegido para la base de datos. Sin embargo, hay aplicaciones más poderosas que soportan diferentes DBMS, estas son útiles ya que tenemos mayor variedad de opciones.

Un visualizador de base de datos que soporta múltiples DBMS es DBeaver [22]. Es una herramienta muy poderosa ya que nos permite visualizar datos, crear y ejecutar *scripts* SQL en diferentes bases de datos, visualizar información detallada de las tablas y diagramas de entidad-relación.

Herramientas para probar APIs

Existen herramientas que nos ayudan a probar las APIs que creamos. Estas nos ayudan a visualizar fácilmente las respuestas de las peticiones, los *headers*, entre otros. Su mayor utilidad es la de permitirnos probar estas APIs antes de integrarlas o usarlas en otros sistemas.

Existen algunas herramientas, por ejemplo Visual Studio Code tiene extensiones básicas que nos permiten probar las APIs de esta forma. Y hay herramientas muy completas como Postman [85] que agregan mayor funcionalidad como la opción de tener diferentes ambientes o generar documentación de la API.

Canales de comunicación de equipo de desarrollo

La comunicación es clave y de mucha importancia en un equipo de trabajo. Además, la elección de una correcta herramienta puede simplificar la comunicación de avances en el equipo. Por ejemplo, algunos de los sistemas mencionados anteriormente se pueden integrar con servicios de comunicación y se podría comunicar por este medio avances en tareas, reporte de errores y fallos y otros. A continuación se listan algunas opciones.

Telegram [106] provee herramientas para integrar con sistemas y recibir notificaciones en tiempo real. La aplicación en si puede salir del contexto de trabajo.

Microsoft Teams [67] es un ambiente para empresas que provee Microsoft. Sin embargo, es un ambiente genérico, no destinado al desarrollo de *software* específicamente.

Slack [101] es una herramienta de comunicación en tiempo real que permite tener diferentes canales para diferentes contexto y provee múltiples herramientas para integrar con diferentes sistemas.

7.1.2. Implementación

El proyecto inició con varias configuraciones realizadas para organizar y planificar el mismo. El proyecto inicia con la implementación de un sistema de control de versiones en donde se seleccionó git con el gestor de GitHub en donde se creó el repositorio que almacena todo el código desarrollado. En la plataforma de GitHub se configuró un proyecto que es un gestor de tareas con dos vistas personalizadas, una tipo tablero y otra tipo lista, en donde las tareas se organizan en tres rubros: *Backlog*, *to do*, *in progress* y *done*. De esta manera todo el equipo pudo estar actualizado sobre que esta trabajando el resto de los integrantes así como sus tareas por completar. Las tareas se manejaron por medio de los *issues* de GitHub en donde se crearon dos plantillas, una de solicitud de una función nueva y otra de reporte de un error. Se configuró de tal manera que al existir un nuevo *issue*, este se colocaba en el rubro *To do* de nuestro gestor de tareas. Posteriormente, se integró GitHub con la plataforma de Slack, una plataforma de comunicación, por medio de Zapier en la cual se configuraron tres automatizaciones. Las automatizaciones consistieron en el envió de un mensaje automático cuando en GitHub se creaba un nuevo *issue*, *pull request* y *merge* a la rama principal.

Posteriormente se activó una GitHub Discussions en donde se discutieron aspectos del proyecto y se crearon canales por tema. Estas discusiones son publicas para que cualquier persona interesada pueda interactuar con el equipo y pueda hacer sus preguntas o incluso contribuir en un futuro.

Luego de la investigación previa de las diferentes tecnologías se decidió continuar el proyecto con las que se detallan a continuación.

El ambiente de desarrollo como un todo contempla el uso de git y se definió un estándar de *Conventional Commits* que consiste de:

- ***feat***: para describir que se están agregando nuevas funcionalidades.
- ***fix***: para describir que se está solucionando algún error reportado.
- ***docs***: para describir que se está agregando documentación.
- ***test***: para describir que se están agregando *tests*.
- **ci**: para describir cualquier modificación en los flujos de integración continua.
- ***refactor***: para describir que no se está modificando la funcionalidad, únicamente se esta cambiando la forma del código.
- ***build***: para describir que se está actualizando los archivos compilados.

Además, se estableció que en el código debe usarse un indentación de cuatro espacios en todos los archivos, que la definición de variables y funciones debe ser semántica para evitar comentarios innecesarios de descripción de variables y funciones. Los comentarios en código quedan a discreción del programador para describir funcionalidades complejas que deben ser detalladas para comprenderlas. Adicionalmente, este trabajo escrito se redacta como parte de la documentación general del código.

Se utilizó Visual Studio Code como editor de código fuente ya que provee las funcionalidades necesarias requeridas, nos provee una integración adecuada con git y acceso a múltiples extensiones que nos servirán en todo el proyecto, como integración con herramientas de *blockchain*.

Para solventar problemas *debuggeando* se escogieron las herramientas DBeaver para la base de datos y Postman para el API. Ambas herramientas son muy poderosas y nos habilitan trabajar fácilmente y resolver problemas con mayor rapidez.

Se decidió trabajar con GitHub como servicio en nube para control de versiones porque este provee múltiples herramientas que se integran nativamente entre ellas y son fáciles de usar. Entre estas herramientas se cuenta con configuración de repositorios privados gratuita, configuración de ramas, GitHub Projects, GitHub Issues, GitHub Discussions, Github Actions, Github Action Secrets y Github Apps. Entonces nos permite utilizar el mismo servicio de control de versiones con las herramientas de seguimiento de requerimientos, integración continua y notificaciones a Slack, que fue la herramienta seleccionada como canal de comunicación. Esto simplifica el trabajo del desarrollador.

7.2. *UI/UX*

7.2.1. *User persona*

Para entender que necesidades son las que se están satisfaciendo, se realizaron dos *user persona* en donde se modelaron usuarios arquetípicos describiendo sus objetivos, frustraciones y su personalidad. Se completó la información según investigaciones realizadas en internet de usuarios que utilizan aplicaciones de venta y compra de viviendas.

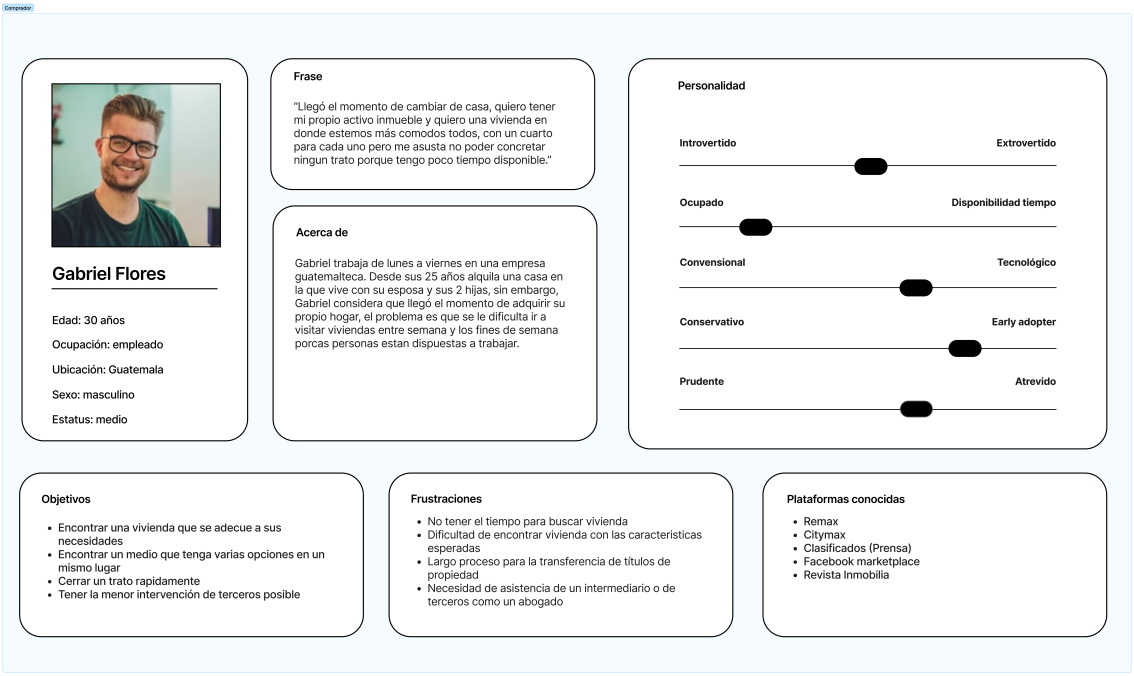


Figura 7.1: *User persona*: comprador

Estos *user persona* se realizaron con el fin de conocer a los usuarios y poder identificar que cosas son las que les gusta de un producto que existe en la actualidad y cuales son las que les frustra. Además, permitió tener simpatía con el público objetivo. Los dos *user persona* creados abarcaron a personas con necesidad de comprar una vivienda y personas con necesidad de vender una vivienda. Los user personas fueron el punto de partida para poder continuar con el resto de pasos relacionados a *UI/UX*.

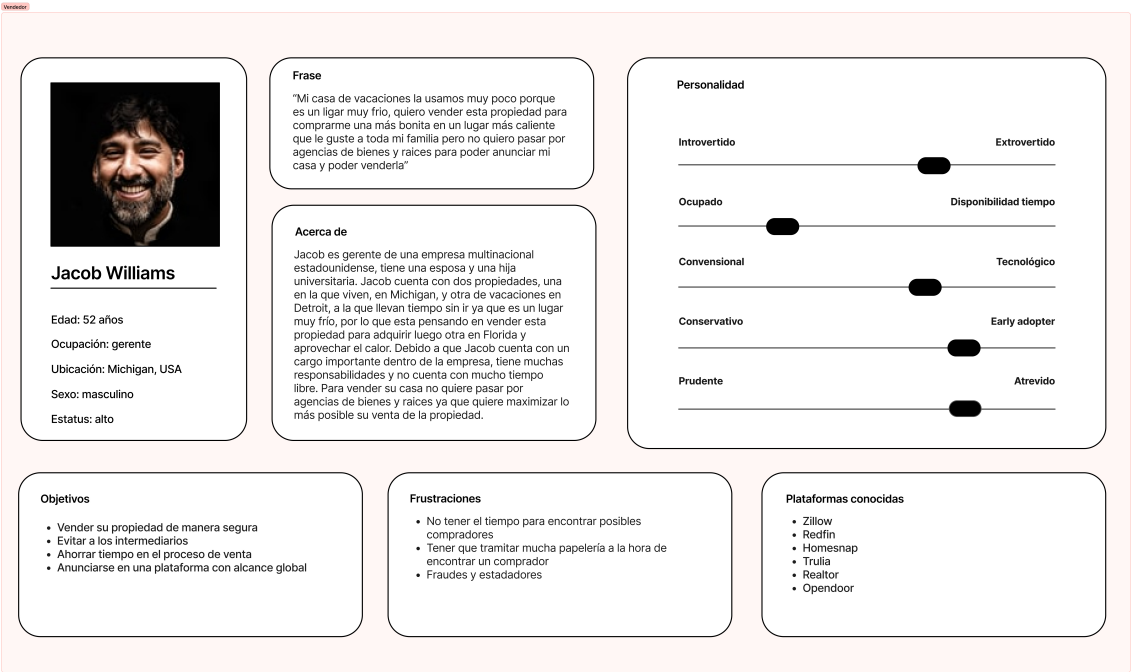


Figura 7.2: *User persona*: vendedor

7.2.2. *User journey*

Con los *user persona* definidos, se realizaron dos *user journey*, uno para cada *user persona*. El *user journey* se dividió en cuatro partes: objetivo o problema del usuario, acciones del usuario, sentimiento del usuario, y las oportunidades. Con estas partes se logró mostrar la interacción de cada uno de los usuarios con cada una de las partes que conforma el sistema que se realizaría para el proyecto para poder identificar la secuencia de acciones que realizaría un usuario, así como desarrollar empatía para mostrar expectativas, comportamientos, emociones, nivel de satisfacción, en cada interacción que tiene el usuario con la plataforma.

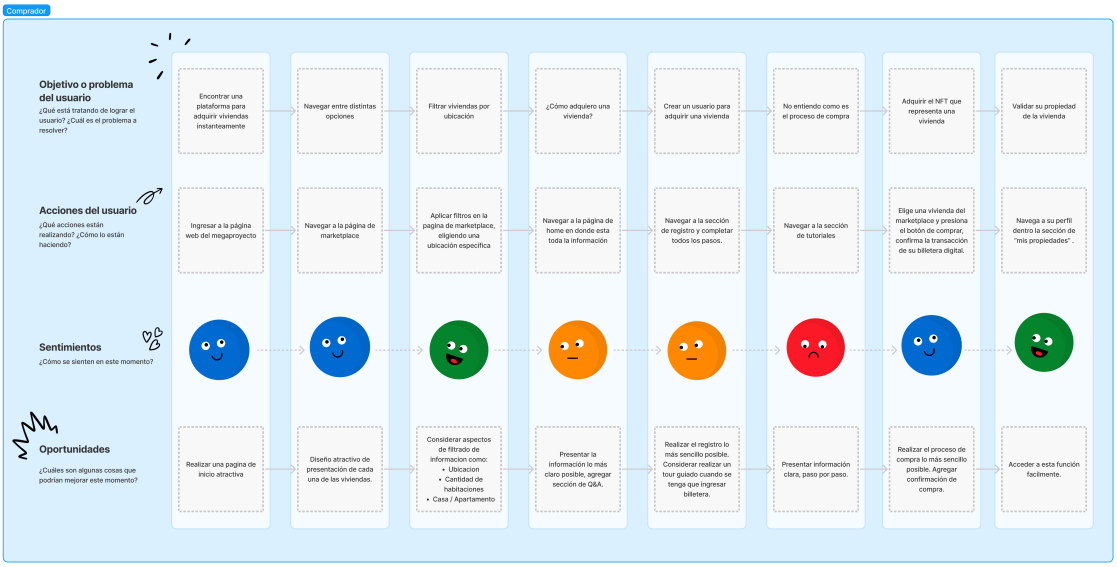


Figura 7.3: *User journey*: comprador

Los *user journey* fueron de utilidad para determinar los requisitos que debe de tener la plataforma y poder identificar los momentos que pueden ser confusos para el usuario y poder simplificarlos.

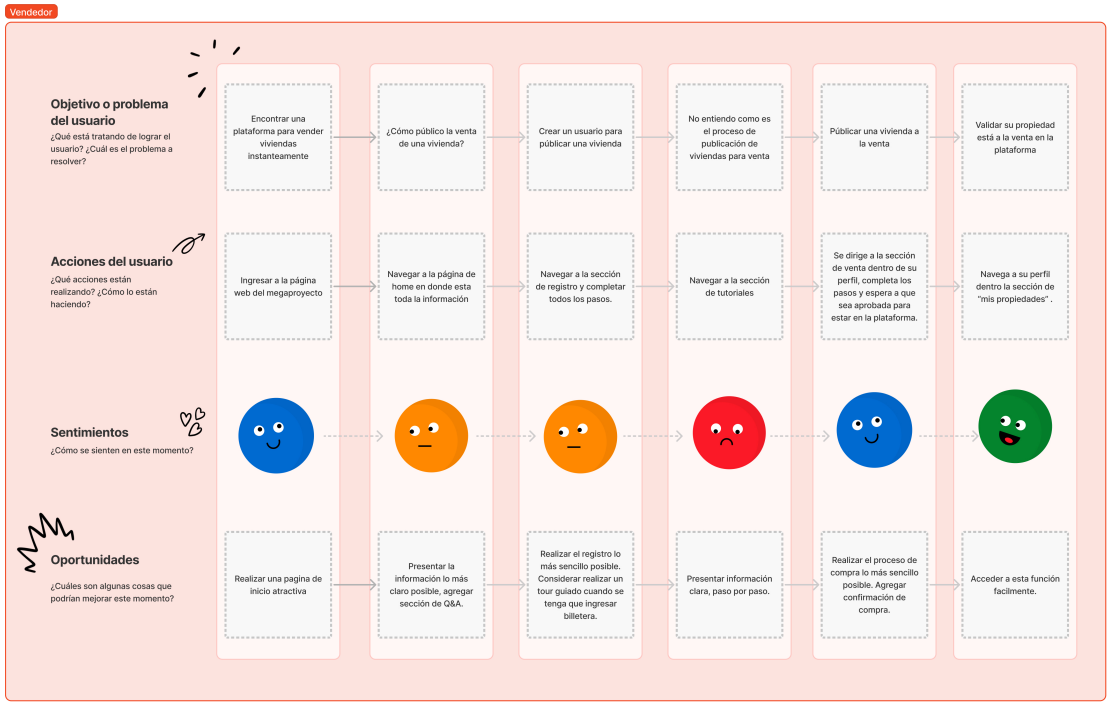


Figura 7.4: *User journey*: vendedor

7.2.3. *Information architecture*

Se realizó la arquitectura de la información con base a los *user journey*, en donde se determinaron, primero, la cantidad de páginas por las que el sitio *web* se debe componer, así como las secciones que cada página debe tener para que el proceso sea intuitivo, sin dejar espacios abiertos a la interpretación del usuario, determinando la organización y creando flujos de navegación adecuados. Esto fue de utilidad para generar una arquitectura del sitio *web* en donde todo lo que el usuario desea, se fácil de encontrar y reducir la posibilidad que usuarios abandonen la plataforma por una navegación complicada.

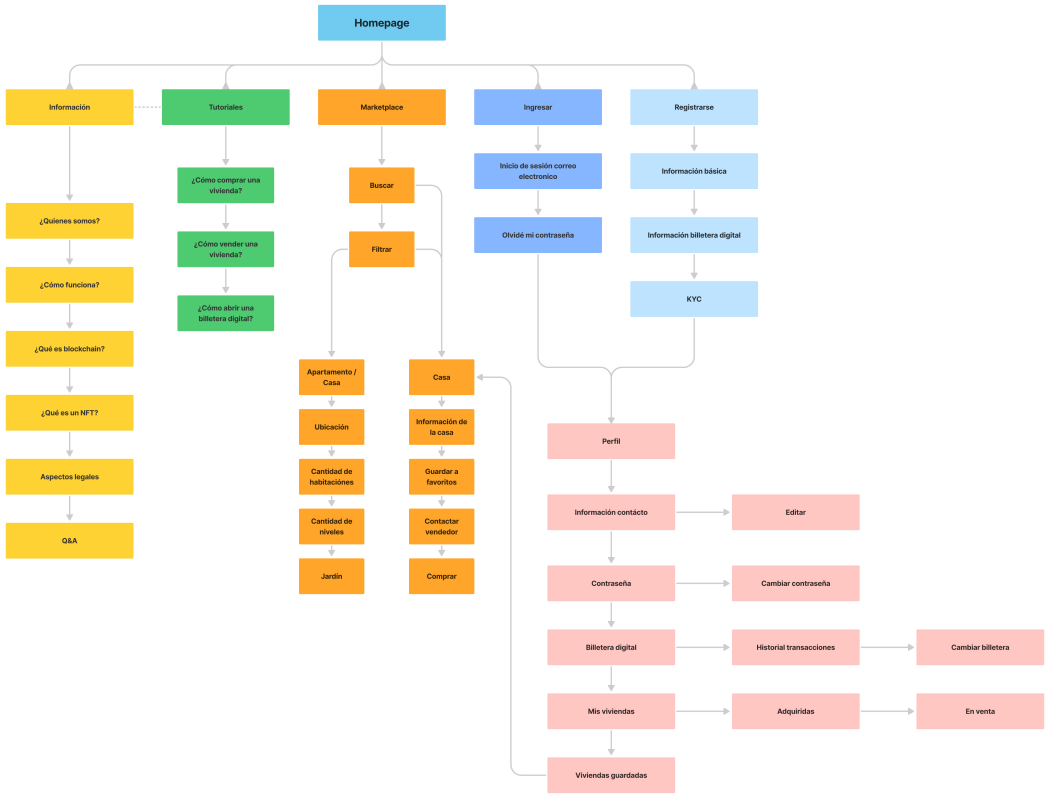


Figura 7.5: Arquitectura de la información

7.2.4. Prototipo

Con la arquitectura de la información definida, se realizó un prototipo de baja fidelidad en donde se estructuró todo el sitio *web*, con espacios para títulos, texto, imágenes y componentes, sin ser elaborados y sin ser texto real ya que su objetivo fue el de validar que la navegación y la estructura del sitio es la adecuada. Una vez se validó este prototipo con los miembros del equipo, se creo una paleta de colores que sería utilizada en el prototipo de alta fidelidad. De manera que, una vez definida la paleta de colores, se realizó el prototipo final, con elementos visuales, y texto real, así como todas las páginas que forman parte del sitio. Este prototipo se realizó con funcionalidad interactiva,

para que se pueda navegar como si fuera el sitio *web* final, de tal manera que pueda ser traducido directamente a código.

7.3. *Smart contract*

7.3.1. Investigación

El desarrollo de *blockchain* inició por la fase de investigación de diversos temas. El primer tema se relacionó a las redes existentes que soportan *smart contracts*. Se realizó una investigación en donde se desarrollaron ocho redes. Para cada red se plasmó el propósito de la misma así como datos interesantes que han ocurrido desde sus inicios. Por medio de esta investigación se realizó una tabla comparativa en donde se compararon temas de transacciones por segundo, mecanismo de consenso y el lenguaje de programación utilizado para desarrollar *smart contract*. Posteriormente se realizó una encuesta en la comunidad de GitHub en donde se compartió esta investigación así como otros recursos valiosos para que entre todos los integrantes decidiéramos la red que utilizaríamos. En este canal de la comunidad se discutió y decidió que la red a utilizar sería Ethereum ya que, en ese momento, estaba pasando por una actualización importante en donde se cambió su mecanismo de consenso de *Proof of Work* a *Proof of Stake* que era una de las debilidades de Ethereum debido al impacto ambiental que tenía. Sin embargo gracias al "*Merge*", se consideró que Ethereum es la red más conveniente ya que es la más grande en cuanto a uso de *smart contracts* y su comunidad es la más grande en relación a las otras. También, históricamente, Ethereum ha sido una red muy estable y se desarrolla por medio del lenguaje de programación de Solidity con el cual ya existe una buena cantidad de documentación.

Una vez se decidió que sería Ethereum la red a utilizar, se realizaron varias investigaciones propias para utilizar las herramientas de vanguardia que el ecosistema ofrece. La primera investigación en este rubro fue acerca de los entornos de desarrollo de Ethereum, en donde se pueda desarrollar, compilar, realizar pruebas y distribuir un *smart contract*. Se utilizó el entorno de Hardhat ya que es uno de los mas nuevos y ofrece una documentación ordenada y robusta. También, tiene integrado la creación de redes de Ethereum sin tener que instalar otro entorno más. Posterior a esta investigación se investigó a profundidad el lenguaje de programación de Solidity para que a la hora de iniciar el desarrollo del *smart contract* tomara el menor tiempo posible. Gracias a esta investigación se descubrieron plataformas que serían útiles para el desarrollo del proyecto. Una de estas fue OpenZeppelin, una plataforma que proporciona diversas funcionalidades de seguridad adaptada para el desarrollo de smart contracts que proporciona librerías de código auditadas con los estándares ERC de Ethereum.

7.3.2. Diseño

Antes de iniciar a desarrollar el *smart contract* se diseñó el mismo con un diagrama sencillo de UML ya que Solidity es un lenguaje de programación orientado a objetos. Este paso se realizó para plasmar el comportamiento deseado del *smart contract* y poder estructurarlo por cantidad de contratos así como las funciones pertenecientes a cada uno y como estos se relacionan.

Figura 7.6: Diseño del *smart contract*

7.3.3. Desarrollo

El desarrollo del smart contract inició con la instalación de todas las herramientas necesarias que se investigaron, de manera que se instaló Hardhat con todas sus dependencias y OpenZeppelin. Una vez instalado Hardhat, se ejecutó y se creo un proyecto nuevo desde cero y se instaló OpenZeppelin.

Una vez instaladas todas las dependencias necesarias, se procedió con el desarrollo del *smart contract* principal y se inició con la implementación del estándar ERC-721, el estándar para la implementación de tokens no fungibles o *NFTs*, para utilizarlo como base para el desarrollo y tener las mejores prácticas de seguridad desde la base del contrato. Dentro de la configuración del ERC-721

se configuró el nombre del token ("Vesta"), así como su símbolo representativo ("V") que son utilizados para listarlo en diferentes plataformas, como *marketplaces* de *NFTs*. Luego, se implementó *ERC721Enumerable*, una interfaz que provee funcionalidad de verificación de pertenencia de *NFTs* por un usuario específico, así como los *NFTs* que han sido creado según un *smart contract*. Esta implementación es necesaria ya que a la hora de conectarlo a una plataforma *web*, se facilita la obtención de los *NFTs* que se han creado y mostrarlos a los usuario.

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";

contract Vesta is ERC721, ERC721Enumerable {
    constructor() ERC721("Vesta", "V") {}
}
```

Script 7.1: (Solidity) Implementación de ERC-721 y definición de token

Luego, se implementó la función encargada de realizar un "*mint*"de *NFT*, que se refiere al proceso en el cual se crea un *NFT* y se almacena en la *blockchain*. En la implementación del *mint* se agregó la funcionalidad de generar un identificador único por *NFT* generado, así como, que al momento de realizar un *NFT*, automáticamente se transfiera a la persona que esta realizando este proceso. Es importante mencionar que no se configuró un suministro máximo ya que de ser así, se podrían tokenizar solo un número limitado de bienes inmuebles, cuando el objetivo es que todos los bienes inmuebles puedan ser tokenizados.

```
function mint() public returns(uint256) {
    uint256 tokenId = _tokenIdCounter.current();
    _tokenIdCounter.increment();
    _safeMint(msg.sender, tokenId);

    return tokenId;
}
```

Script 7.2: (Solidity) Implementación método *mint*

Una vez se implementaron las funciones necesarias para el funcionamiento básico de un *NFT*, se implemento la función que se encarga de agregar los metadatos que son en donde se definen los atributos del mismo como el nombre, la imagen, descripción, etc. Esta implementación se realizó respetando el esquema del estándar ERC-721 publicado en los archivos de la fundación Ethereum. Para lograrlo, se realizó una función llamada *tokenURI* encargada de retornar una URL que apunta a un archivo JSON que contiene la información de los metadatos. La URL retornada es realizada dentro de *blockchain* atreves de una URL codificada en base 64 utilizando el estándar de internet de *Data URL* ya que estas URL están diseñadas para que dentro la misma URL este el contenido que regresa, sin necesidad de realizar una petición HTTP hacia un archivo externo.

```
function tokenURI(uint256 tokenId) public view override returns(string memory) {
    require(
        _exists(tokenId),
        "ERC721 Metadata: URI query for nonexistent token"
    );

    (string memory a, string memory aa, uint256 b, uint256 c, uint256 d, uint256 e,
    uint256 f, uint256 g, bool h, bool i, string memory j) = get_house(tokenId);

    string memory jsonURI = Base64.encode(
        bytes(
            string(
                abi.encodePacked(
```

```
                    '{"name":"', "Vesta #",
                    tokenId.toString(),
                    '","description":"', "Vesta is a property tokenization platform,
    represented as NFTs that live on the blockchain for easy transfer of ownership.",
                    '","image":"', a,
                    '","attributes":[{"trait_type":"Typology","value":"', aa, '"},{"
    trait_type": "Year built","value":"', b.toString(), '"},{"trait_type": "Square
    meters","value":"', c.toString(), '"},{"trait_type": "Rooms","value":"', d.
    toString(), '"},{"trait_type": "Bathrooms","value":"', e.toString(), '"},{"
    trait_type": "Levels","value":"', f.toString(), '"},{"trait_type": "Parkings","
    value":"', g.toString(), '"},{"trait_type": "Yard","value":"',  boolToString(h),
    '"},{"trait_type": "Pool","value":"',  boolToString(i), '"},{"trait_type": "
    Location","value":"', j, '"}]}'
                )
            )
        )
    );

    return string(abi.encodePacked("data:application/json;base64,", jsonURI));
}
```

Script 7.3: (Solidity) Implementación método *tokenURI*

Una vez se implementaron los metadatos del *NFT*, se creó un nuevo *smart contract* destinado a la toma de datos de las especificaciones del bien inmueble por medio del usuario. Para lograr esto, se creo una estructura de datos que tiene como propiedades todas las especificaciones de una casa como ubicación, metros cuadrados y año de construcción, así como número de habitaciones, baños, cantidad de niveles, etc.

```
struct House{
    string image;
    string typology;
    uint256 yearBuilt;
    uint256 sqm;
    uint256 rooms;
    uint256 bathrooms;
    uint256 levels;
    uint256 parkings;
    bool yard;
    bool pool;
    string location;
}
```

Script 7.4: (Solidity) Estrucutra de datos en representación a un inmueble

Posteriormente, se crearon dos funciones, una destinada a la escritura de todas las propiedades de una casa y otra destinada a la lectura. De manera que este *smart contract* se realizó con el fin de escribir y consultar todas las especificaciones de algún bien inmueble para luego transferirlo a los metadatos del *NFT*. Es por esto, que luego se implementó este contrato en el contrato principal y se almacenaron todos los datos en los metadatos. Con esto finalizado, se creó toda la funcionalidad del *smart contract* para poder tokenizar un bien inmueble.

7.3.4. *Testing*

Con el *smart contract* funcional, se procedió a la realización de pruebas para verificar que se este comportando de la manera deseada. Las pruebas se realizaron con el lenguaje de programación de JavaScript y el *framework* de Chai. Se describió un solo caso de uso con seis pruebas independientes. Las pruebas, a su vez, se dividieron en tres rubros: *deployment*, *minting* y *token uri*. La prueba realizada en *deployment* fue precisamente que el contrato se despliegue en la red. Las pruebas realizadas

en el rubro de *minting* fueron: asignar automáticamente el dueño del *NFT* a la hora de realizar un *mint*, verificación de que el *smart contract* no asigne un suministro máximo, incrementar el balance por +1 por cada *mint* realizado, y transferir *NFTs* entre diferente cuentas. La prueba realizada en *token uri* consistió en la verificación de que los metadatos producidos respeten el estándar de metadatos propuesto por la fundación Ethereum en su estándar ERC-721.

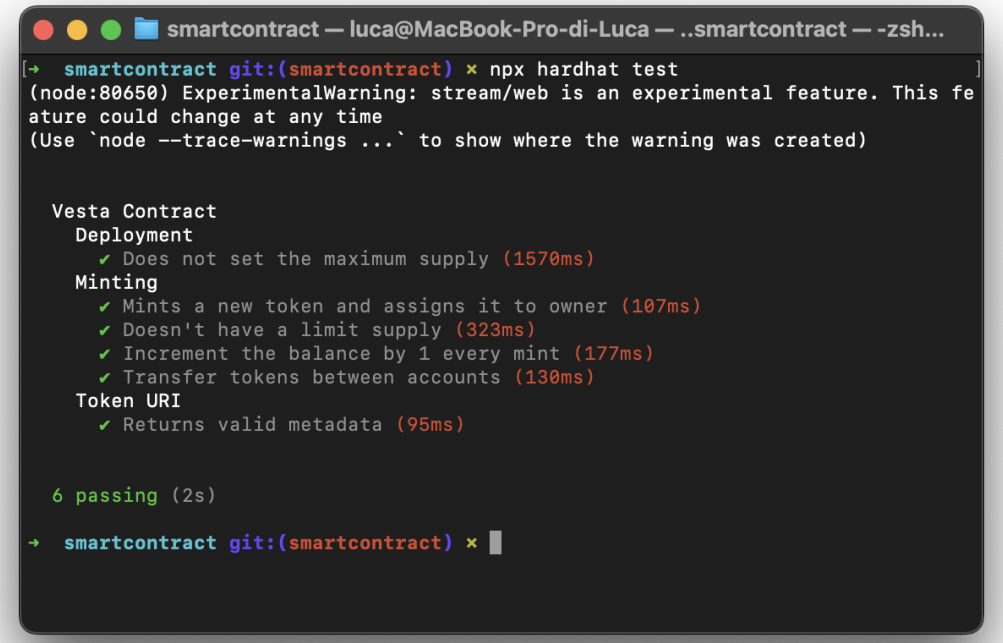


Figura 7.7: Resultado *testing smart contract*

También, se analizaron los contratos por medio de la plataforma de Mythril, una herramienta de análisis de seguridad para *smart contracts* de Ethereum. Mythril detecta una variedad de problemas de seguridad, específicamente vulnerabilidades comunes.

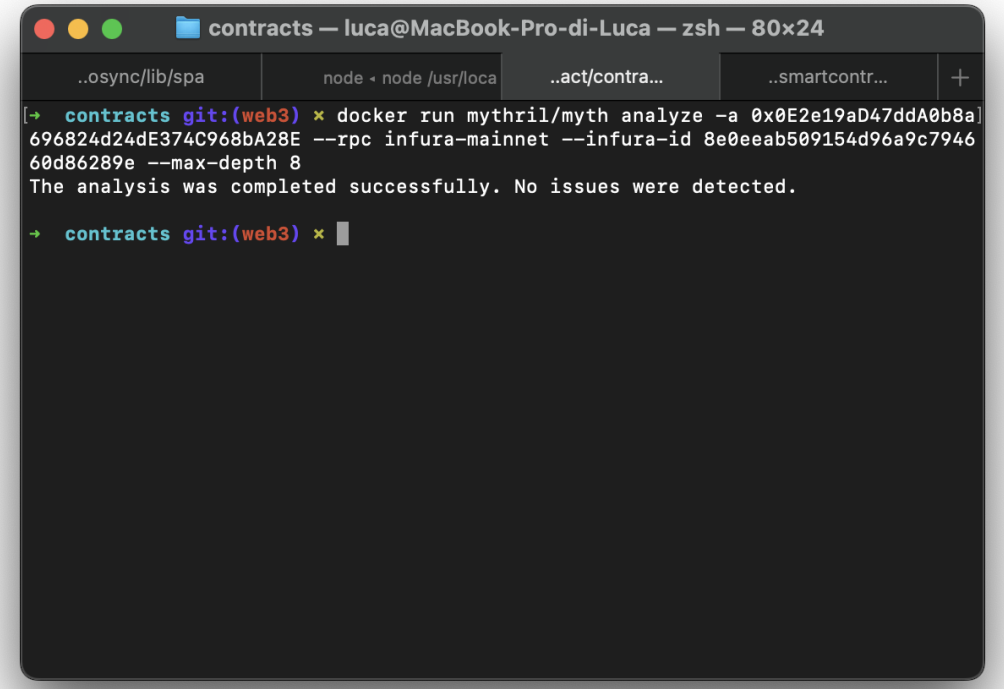


Figura 7.8: Análisis de los *smart contracts* con Mythril

7.3.5. *Deploy*

Se configuró el ambiente por medio de la configuración de Hardhat para desplegar el *smart contract* a una red de prueba de Ethereum, en este caso, Görli. Para esto, se configuró una transacción en donde el contenido de la misma fueron los datos del *smart contract* compilado.

```
const deploy = async () => {
    const [deployer] = await ethers.getSigners();

    console.log("Deploying contract with the account: ", deployer.address);

    const Vesta = await ethers.getContractFactory("Vesta");
    const deployed = await Vesta.deploy();

    console.log("Vesta is deployed at: ", deployed.address);
}

deploy()
    .then(() => process.exit(0))
    .catch((error) => {
        console.log(error)
        process.exit(1)
    });
```

Script 7.5: (JavaScript) Transacción para desplegar un *smart contract*

Esta transacción se envió a un nodo de la red de Görli de Ethereum por medio de la creación de una instancia de Infura que es un proveedor de infraestructura como servicio. Esta instancia se agrego al archivo de configuración de Hardhat con variables de entorno para mantener la seguridad de las llaves privadas y no estén disponibles en el repositorio de GitHub. Adicionalmente, se agregó la mainnet de Ethereum y la Avalanche Fuji *testnet* para desplegar en estas redes.

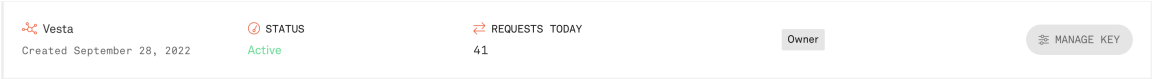


Figura 7.9: Instancia de Infura

```
require("@nomicfoundation/hardhat-toolbox");
require('dotenv').config();

const projectId = process.env.INFURA_PROJECT_ID
const privateKey = process.env.DEPLOYER_SIGNER_PRIVATE_KEY


module.exports = {
  solidity: "0.8.17",
  networks: {
    goerli: {
      url: `https://goerli.infura.io/v3/${projectId}`,
      accounts: [privateKey],
    },
    mainnet: {
      url: `https://mainnet.infura.io/v3/${projectId}`,
      accounts: [privateKey],
    },
    fuji: {
      url: `https://avalanche-fuji.infura.io/v3/${projectId}`,
      accounts: [privateKey],
    }
  },
};
```

Script 7.6: (JavaScript) Configuración para desplegar el *smart contract* en *testnets* y *mainnet*

Luego, se ejecutó esta transacción y se le dio seguimiento en Etherscan, un explorador de bloques. Cuando la transacción fue validada, se verificó y publico el contrato en Etherscan por medio de la publicación del código desarrollado previamente en el *smart contract*, sin embargo, para verificarlo es necesario que exista un solo archivo. Este archivo se generó por medio de la función de *flatten* que provee Hardhat. Una vez Etherscan verificó que el código que se subió coincide con el código del *smart contract* de *blockchain*, el contrato se validó en la red y dio acceso a varias funcionalidades. Una de estas funcionalidades fue la de poder interactuar con el contrato desde su misma plataforma, de manera que se interactuó con esta interfaz para crear el primer *NFT* del contrato y validar que el contrato este realizando todas las funciones de manera correcta. Una vez se interactuó con el contrato, fue posible observar el *NFT* en todas las plataformas de *marketplace* de *NFTs* más famosas del mercado como OpenSea y Rarible. Con la visualización del *NFT* en estas plataformas se pudo verificar que los metadatos se hayan creado de manera exitosa ya que todas las plataformas fueron capaces de interpretarlos y desplegarlos en cada una de sus interfaces de usuario.

7.3.6. Monitoreo

Una vez el contrato se desplegó en la red, se implementó Tenderly, una plataforma que permite probar y monitorear *smart contracts* que se encuentran en la red. En Tenderly se crearon 3 alarmas que se conectaron al canal de Slack. Una alarma para cuando sucede una transacción fallida, cuando hay una transacción exitosa y cuando se ejecuta la función e transferencia de propiedad (Ver anexo D.1 para ver las alertas activas).

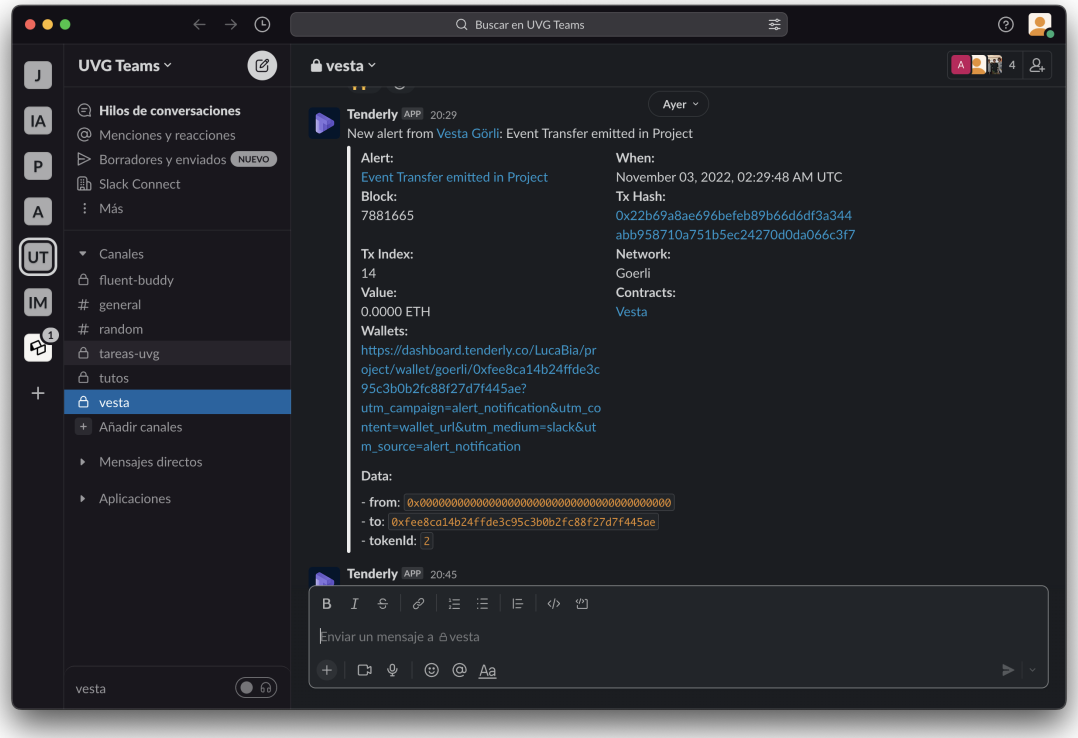


Figura 7.10: Notificaciones de alerta de interacción con el *smart contract*

También se configuraron simulaciones en donde se probaron diferentes escenarios para poder replicar ciertos escenarios. Estas simulaciones actúan como si se estuviera interactuando de manera real con el *smart contract* (Ver anexo D para ver simulaciones detalladas).

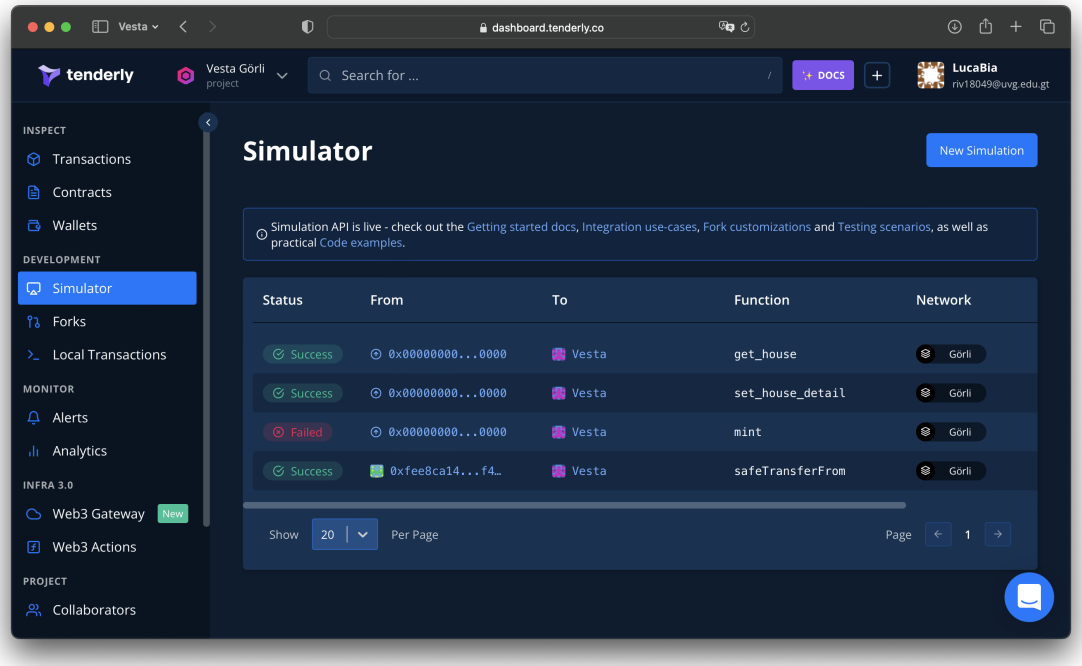


Figura 7.11: Simulaciones de transacciones

7.4. *Backend*

El *backend* suele ser el punto principal de lógica en una aplicación *web*. Este usualmente es la única fuente de la verdad en un sistema, ya que tiene acceso a la base de datos y se encarga de procesar datos antes de exponerlos en el *frontend* o en un API. Mas en este proyecto, el propósito del *backend* no es el del rol principal. La red de *blockchain* que complementa este proyecto actúa como base de datos principal; esto supone un reto crucial, el cual se puede plantear de la siguiente manera: "¿Cómo debe interactuar el *backend* con el *blockchain* y el *frontend*?".

En este caso, el *backend* cumple el papel de información adicional a la red. Este nos ayudará a almacenar información, de las entidades en el sistema, que tiene un costo elevado el almacenarlas en la red como documentos e imágenes y otros datos que vale la pena almacenarlos en un sistema tradicional, como información de los usuarios en la red. *Blockchain* tiene la característica de ser anónimo, los usuarios en la red son representados por *Wallets* y estos no suelen estar ligados a una identificación o nombre, además un solo usuario puede poseer varias *wallets* a gusto. Así que almacenaremos imágenes y datos adicionales de los usuarios que son parte de la red.

7.4.1. Diseño

Tecnologías

Existe una gran cantidad de opciones de *frameworks web* para *backend*. Estos *frameworks* facilitan el desarrollo de aplicaciones ya que contienen módulos y herramientas útiles que reducen o eliminan la cantidad de tareas repetitivas durante el desarrollo de la aplicación. Al elegir un *framework* también se debe contemplar que pueden estar escritos en diferentes lenguajes de programación y esto influye por el hecho de que hay lenguajes de programación que tienen otras librerías útiles que podríamos llegar a necesitar más adelante. También hay que considerar la estructura de archivos y otras herramientas que nos provean. A continuación se listan los *frameworks* que se tomaron en cuenta al momento de decisión de la tecnología.

- **Django** [24]
- **Ruby on Rails** [94]
- **Express JS** [31]
- **Laravel** [62]

Se decidió descartar Express JS, porque este *framework* no provee una estructura de archivos base, lo que puede ocasionar un mal uso del *framework* y desorganización. También se decidió descartar Laravel, ya que supondría un reto innecesario para el propósito de este trabajo aprender PHP y el *framework* desde cero. Y entre Django y Ruby on Rails, se optó por usar Ruby on Rails ya que la estructura de archivos parece tener mayor facilidad para escalar fácilmente en caso de ser necesario y provee soporte nativo para servicios de almacenamiento en nube.

Cualquier variación en la selección de cualquiera de todas las tecnologías mencionadas anteriormente no deberían de afectar ni limitar el resultado final y no afecta al propósito de este trabajo que es el de presentar una propuesta de *backend* para conectar con *blockchain* y *Web3*. Así que se pudo haber llegado al resultado final sin la necesidad de escoger las mismas tecnologías por las que se optaron.

7.4.2. Base de datos

Tecnologías

En cuanto nos referimos a la base de datos también existe una gran variedad de opciones para escoger tecnologías. Principalmente se debe seleccionar el tipo de base de datos, hablando de relacionales y no relacionales. Las base de datos no relacionales nos ofrecen facilidad de escalabilidad, facilidad en la implementación de algoritmos de busca y recomendaciones, entre otros. Sin embargo, estas características no son de gran utilidad para la implementación deseada, por lo tanto que se decidió optar por una base de datos relacional.

Entre las principales opciones que se consideraron están MySQL [77] y PostgreSQL [84]. Ambas base de datos son buenas opciones ya que MySQL es de lectura rápida y fácil mantenimiento, y PostgreSQL es una base de datos igualmente rápida en lectura y escritura y es más poderosa, ya que se desempeña muy bien con *queries* complejos. Para la elaboración de este proyecto no se necesita

una base de datos con características específicas ya que no se almacenará información en grandes volúmenes.

Se decidió optar por PostgreSQL ya que se tiene un mayor conocimiento sobre este y en caso de ser necesario almacenar más información y escribir y leer con mayor rapidez por las solicitudes del *blockchain* no sería necesaria una migración.

Diagrama ER

Para el diseño de la base de datos se esbozó la idea general de todo el proyecto. Este ejercicio fue de gran utilidad para poder visualizar que datos se iban a necesitar para completar los requerimientos planteados.

Se diseñó la base de datos según los requerimientos observados anteriormente. Lo deseado era almacenar información adicional de las entidades en el sistema, *User* y *Property*; esto con el fin de reducir la carga de almacenamiento de datos en la red *blockchain* y así mismo reducir costos de transacciones en la misma red.

Como se mencionó, las entidades principales que conforman la parte más importante de este sistema son *User* y *Property*. Adicionalmente, son relevantes la entidad *Wallet* que representa una cuenta en la red *blockchain* y la entidad *Blob* que se encargara de almacenar los archivos del sistema. Así que se procedió a analizar que campos debían almacenarse en base de datos.

User

- (*String*) ***email*** será utilizado para identificar al usuario en la base de datos.
- (*String*) ***password*** será utilizado para que el usuario pueda iniciar sesión a la plataforma utilizando su *password*.
- (*Boolean*) **active** privatizará el acceso a la plataforma en caso de ser necesario.
- (*String*) ***first_ name*** almacenará el nombre del usuario.
- (*String*) ***last_ name*** almacenará el apellido del usuario.
- (*String*) ***telephone*** almacenará el teléfono del usuario.
- (*String*) ***pid_ number*** almacenará el número de identificación personal (de sus siglas en inglés).

Property

- (*String*) ***nft_ id*** será el identificador que tiene esta propiedad en la red *blockchain*.
- (*Integer*) **finca** (información de la propiedad según el registro de propiedad guatemalteco).
- (*Integer*) **folio** (información de la propiedad según el registro de propiedad guatemalteco).
- (*Integer*) **libro** (información de la propiedad según el registro de propiedad guatemalteco).
- (*String*) ***location*** almacenará la ubicación de la propiedad.

- (*String*) ***status*** almacenará el estado de la propiedad y ayudará a definir si está disponible para compra o no, el cual puede ser equivalente a *listed*, *not_ listed* o *reserved*.
- (*String*) ***category*** almacenará la categoría de la propiedad y ayudará a filtrar en búsquedas. Este puede ser equivalente a *house*, *land* o *apartment*, *commercial* u *other*.
- (*Integer*) ***rooms*** almacenará la cantidad de habitaciones de la propiedad.
- (*Integer*) ***bathrooms*** almacenará la cantidad de baños de la propiedad.
- (*Float*) ***latitude*** almacenará la latitud de la propiedad.
- (*Float*) ***longitude*** almacenará la longitud de la propiedad.
- (*Integer*) ***price*** almacenará el precio de la propiedad.

Wallet

- (*String*) ***account*** almacenará la dirección que representa la *wallet* del usuario. Un usuario puede tener múltiples *wallets* asociadas y puede iniciar sesión con cada una de ellas.

Considerando que se tomará ventaja de la integración nativa de almacenamiento que provee Ruby on Rails, no es necesario diseñar la base de datos, ya que se administra por *ActiveStorage*. Adicionalmente, El módulo *ActiveRecord* de Ruby on Rails gestiona por nosotros los *ids* de las tablas y coloca *timestamps* que pueden llegar a ser útiles.

Se decidió colocar *User* y *Property* como entidades principales ya que son independientes, a diferencia de *Wallet* la cual no puede existir si no hay un *User*. Así que se colocó *Wallet* dentro del *scope* de *User*. Luego del análisis previo se realizó el diagrama de entidad-relación como se puede ver a continuación.

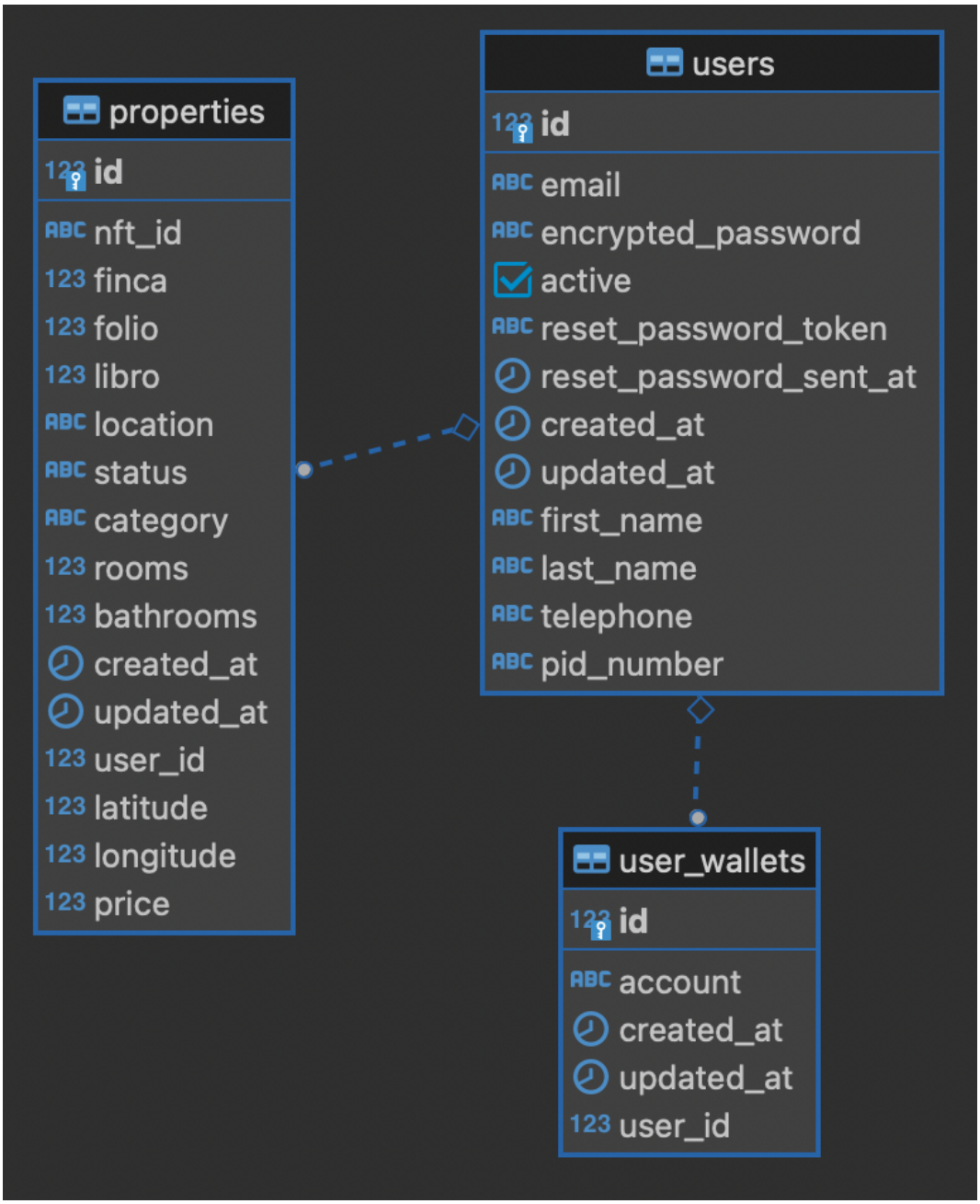


Figura 7.12: Diagrama de entidad-relación.

7.4.3. Implementación

A continuación se trata el proceso de implementación que se siguió para el desarrollo del *backend*. Se utilizó Ruby on Rails por la madurez del *framework* y la gran cantidad de herramientas que provee para el desarrollo *web* y su entorno completo.

Configuración

Para instalar adecuadamente Ruby se instaló y configuró RVM [97]. Este nos permite manejar con mayor facilidad las versiones disponibles de Ruby. La instalación de RVM es sencilla, pero puede tomar un tiempo considerable. Los pasos de instalación se pueden encontrar en su página *web*. Consiste en la instalación de las llaves GPG y luego la instalación de RVM.[1] Con RVM se pueden usar los comandos *list*, *install* y *use* para cambiar entre diversas versiones de Ruby.

Luego de instalar Ruby se pudo continuar con el proceso de creación del proyecto de Rails [Ver Anexo F.1] que se detalla a continuación.

```bash
# Revision de la version de Ruby
ruby --version

# Instalacion de Rails
gem install rails

# Revision de la version de Rails
rails --version

# Creacion de un proyecto de Rails
rails new realestate-nft-marketplace

# Ingresar al directorio creado
cd realestate-nft-marketplace

# Ejecutar el servidor por primera vez
rails s
```

Script 7.7: (Bash) Instalación de Rails

El siguiente paso consiste en la instalación de las gemas.[2] Estas gemas se especifican en un archivo llamado *Gemfile*, en el cual se modificó la línea donde se especifica la gema que se utilizará para conectarse a la base de datos. Por defecto, Rails instala la gema sqlite3 para usar como gestor de base de datos. Sin embargo, se decidió optar por PostgreSQL [82], como se mencionó con anterioridad en este documento.

Para lograr esto se reemplazó la siguiente línea de código.

```ruby
# ...

# gem "sqlite3", "~> 1.4"
gem 'pg', '1.3.3'

# ...
```

Script 7.8: (Ruby) Instalación de gema de PostgreSQL

[1] Nota: RVM suele no tener buena compatibilidad con el sistema operativo Windows, pero existen otras herramientas que proveen la misma funcionalidad que se pueden utilizar como alternativa.

[2] Nota: Ruby le llama gemas a sus paquetes de librerías.

Luego de que se modificó el *Gemfile* se ejecutó el comando que lee el archivo e instala las gemas.

```
bundle install

# Al instalar la gema pg puede surgir un error con su instalacion, se soluciono instalando el siguiente paquete.
brew install libpq                  # MacOS
sudo apt-get -y install libpq-dev   # Ubuntu

# Y se ejecuta nuevamente bundle install
```

Script 7.9: (Bash) Instalación de gemas

Al finalizar la instalación se terminó de configurar el ambiente de Ruby on Rails, esto involucra el archivo de credenciales y *environments* donde se configuró todas las credenciales que se usan en la aplicación como contraseña de base de datos, accesos al *bucket* de AWS [10] donde se almacenarán las imágenes, entre otras configuraciones. Todo esto se realizó de la siguiente manera.

Environments y *Credentials*

```
# Se abre el archivo cifrado del ambiente 'development'
EDITOR="nano" rails credentials:edit --environment development

# Se coloca el siguiente contenido con las respectivas credenciales
db:
    port: 5432
    username: postgres
    password: ""
    database: nft_realestate_dev

aws:
    access_key_id: ""
    secret_access_key: ""
    region: ""
    bucket: ""

hmac:
    secret: somehmacsecret

secret_key_base: somesecretkeybase
```

Script 7.10: (Bash) Configuración de *RAILS_ENV* y credenciales

Database

```
# DB configuration

default: &default
    pool: 30
    host: 127.0.0.1
    adapter: postgresql
    encoding: unicode
    reconnect: false

development:
    <<: *default
    port: <%= 5432 %>
    username: <%= Rails.application.credentials.db[:username] %>
    password: <%= Rails.application.credentials.db[:password] %>
    database: <%= Rails.application.credentials.db[:database] %>

test:
    <<: *default
    port: <%= Rails.application.credentials.db[:port] %>
    username: <%= Rails.application.credentials.db[:username] %>
```

```
    password: <%= Rails.application.credentials.db[:password] %>
    database: <%= Rails.application.credentials.db[:database] %>

production:
    <<: *default
    host: <%= Rails.application.credentials.db[:host] %>
    port: <%= Rails.application.credentials.db[:port] %>
    username: <%= Rails.application.credentials.db[:username] %>
    password: <%= Rails.application.credentials.db[:password] %>
    database: <%= Rails.application.credentials.db[:database] %>
```

Script 7.11: Configuración de archivo config/database.yml

Storage

```
test:
    service: Disk
    root: <%= Rails.root.join("tmp/storage") %>

local:
    service: Disk
    root: <%= Rails.root.join("storage") %>

#
aws:
    service: S3
    access_key_id: <%= Rails.application.credentials.dig(:aws, :access_key_id) %>
    secret_access_key: <%= Rails.application.credentials.dig(:aws, :secret_access_key
    ) %>
    region: <%= Rails.application.credentials.dig(:aws, :region) %>
    bucket: <%= Rails.application.credentials.dig(:aws, :bucket) %>
```

Script 7.12: Configuración de archivo config/storage.yml

Production Environment

```
# ...

config.active_storage.service = :aws

# ...
```

Script 7.13: (Ruby) Configuración de archivo config/environments/production.rb

Migrations

Las migraciones son la herramienta que Rails nos provee para modificar el *schema* de la base de datos. Prácticamente es una interfaz que nos permite describir las tablas, columnas y relaciones de nuestra base de datos con código Ruby y esta es traducida a SQL al ejecutarse.

Para generar los archivos de migraciones se ejecutó el comando de creación de *scaffold* de una entidad nueva [Ver Anexo F.2]. Este comando genera varios archivos necesarios en todos los niveles de la estructura MVC para Rails. Se crearon las migraciones con la estructura de base de datos deseada y luego se ejecutaron las migraciones como se muestra a continuación. [Ver Anexo F.3]

```
rails db:create
rails db:migrate
rails db:migrate:status
```

Script 7.14: (Bash) Ejecución de migraciones

Models

Con nuestra base de datos hecha se pudo empezar a trabajar con los modelos. Al ser una aplicación sencilla, en los modelos solo se tuvo que definir asociaciones, validaciones y unos *enums* donde aplicaban. Para el manejo de usuarios se aprovechó la funcionalidad que provee la gema *devise* [23], ya que es una herramienta muy potente y con múltiples soluciones. A continuación, se presenta el código que se implementó en los modelos.

```ruby
class User < ApplicationRecord
    devise :database_authenticatable, :registerable, :recoverable

    has_many :wallets
    has_many :properties

    # Asociacion definida para adjuntar imagen como avatar para los users
    has_one_attached :avatar do |attachable|
        attachable.variant :thumb, resize_to_limit: [150, 150]
    end
end
```

Script 7.15: (Ruby) app/models/user.rb

```ruby
class Property < ApplicationRecord
    belongs_to :user

    enum status: {
        listed: 'listed',
        not_listed: 'not_listed',
        reserved: 'reserved'
    }, _default: 'not_listed'

    enum category: {
        house: 'house',
        land: 'land',
        apartment: 'apartment',
        commercial: 'commercial',
        other: 'other'
    }

    validates :status, inclusion: { in: statuses.keys }
    validates :category, inclusion: { in: categories.keys }

    # Asociacion definida para adjuntar archivos para las properties
    has_many_attached :files
end
```

Script 7.16: (Ruby) app/models/property.rb

```ruby
class User::Wallet < ApplicationRecord
    belongs_to :user
end
```

Script 7.17: (Ruby) app/models/user/wallet.rb

7.4.4. API

Configuración

Para que funcionara adecuadamente el API se configuró un *middleware* para los CORS *Headers*, en Ruby on Rails, se puede configurar fácilmente en el archivo de configuración principal y para esto también fue necesario instalar la gema *rack-cors* [88].

```ruby
require_relative 'boot'

require 'rails/all'

# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(*Rails.groups)

module RealestateNftMarketplace
    class Application < Rails::Application
        # Initialize configuration defaults for originally generated Rails version.
        config.load_defaults 7.0

        # Configuration for the application, engines, and railties goes here.
        #
        # These settings can be overridden in specific environments using the files
        # in config/environments, which are processed later.
        #
        # config.time_zone = "Central Time (US & Canada)"
        # config.eager_load_paths << Rails.root.join("extras")

        config.middleware.insert_before 0, Rack::Cors do
            allow do
                origins '*'
                resource '/api/*',
                    headers: :any,
                    expose: %w[access-token expiry token-type uid client],
                    methods: %i[get post options put]
            end
        end
    end
end
```

Script 7.18: (Ruby) config/application.rb

API *Controllers*

Luego de tener configurado el API, se procedió a crear el *controller* principal, del cual heredarían todos los *controllers* de tipo API. Este *controller* de API principal se encarga de validar la autorización de todas las *request* que se hacen. Se utilizaron JSON *Web Tokens* como método de autenticación y para generarlos se instaló una gema de ruby llamada *ruby-jwt* [93]. A estos tokens se les definió un tiempo de expiración y un *sub*, el cual es el el estándar para mandar el *id* de la entidad que el JWT representa.

```ruby
class ApplicationApiController < ActionController::API
    include ActionController::MimeResponds
    include ApplicationApiHelper

    before_action :authorize_request

    def authorize_request
        token = nil
```

```ruby

        token = request.headers['Authorization'].split('JWT ')[1] if request.headers['Authorization'].present?

        # No token provided
        return respond_with_status(401) unless token

        # Check if token is valid
        valid, payload = AuthToken.verify token

        return respond_with_status(401) unless valid

        # Looking for the user
        @current_user = User.find_by_id(payload['sub'])

        return respond_with_status(401) if @current_user.blank?

        return respond_with_status(401, 'Your user is not active') unless @current_user.active
    end
end
```

Script 7.19: (Ruby) app/controllers/application_api_controller.rb

Entonces, para que este *controller* API principal funcionara correctamente se le incluyó un *helper*, el cual es una clase que tiene método útiles y que estos estuvieran accesibles en todas las APIs, como es el método que responde con un estado HTTP. Adicionalmente se creó el servicio *AuthToken* que es el encargado de interactuar directamente con la gema *ruby-jwt* y añade un nivel de abstracción que simplifica su uso. Luego de hacer el resto de *controllers* del API se expusieron los métodos utilizando el archivo routes.rb. En el apartado de Anexos se puede ver la implementación de los *controllers*, *helper* y servicios [Ver Anexo C1-C4].

7.4.5. *Testing*

El *testing* es fundamental en el ciclo de vida de desarrollo. Este nos permite ofrecer código de mayor calidad ya que gracias al *testing* se puede identificar problemas con anticipación. Adicionalmente, tener una amplia cobertura de *tests* en un sistema puede ayudarnos a reducir significativamente la cantidad de vulnerabilidades al momento de publicarlo. Debido a esto se decidió agregar *testing* sobre las APIs más importantes como las son las de inicio de sesión y registro [Ver Anexo C.5.].

En los *tests* se definieron diferentes casos exitosos y fallidos de *login* y registro. Entre más casos de uso cubre un *test*, es más útil ya que nos ayudan a evaluar diferentes escenarios. Adicionalmente, se realizaron *test* de estrés para probar el API.

7.4.6. CI

CI o Integración Continua es una práctica que promueve la eficiencia de trabajo. Esta permite que los desarrolladores trabajen con mayor rapidez porque se reduce la necesidad de realizar pruebas en cada cambio que hacen. Un flujo de CI adecuado contempla la compilación y ejecución automática del sistema y además la ejecución de test automatizado. Se decidió crear e integrar un flujo de CI en este proyecto ya que el ambiente de desarrollo (GitHub) proporciona herramientas que facilitan la creación de estos flujos.

Los Github Workflows son ejecutados por las Github Actions, se puede crear un flujo de pasos para seguir y especificar instrucciones de instalación, compilación y ejecución de la aplicación y sus *tests*. Considerando que la plataforma tiene credenciales se utilizó la herramienta Github Action Secrets para almacenar las credenciales necesarias para el flujo de instalación y ejecución de sistema.

Luego se definieron todos los pasos para la correcta instalación de la aplicación de Rails y la ejecución de sus *tests* en un *job*. Adicionalmente, se agregó un *job* que ejecuta los *test* desarrollados en otro módulo encargados de realizar pruebas a los *smart contracts* generados. Y finalmente, se agregó un *job* que ejecuta unos *linters* para revisar y validar buenas prácticas en el código con Rubocop [92], posibles vulnerabilidades con las gemas y sus dependencias con Bundler-audit [15] y vulnerabilidades de seguridad con Brakeman [13].

Este *workflow* completo se definió para ejecutarse en cada *push* y *pull request* a las ramas *master* y *production* y prohíbe el *merge* si alguno de los *checks* que se ejecutan falla; obligando al desarrollador a solucionar los problemas indicados antes de continuar [Ver Anexo C.6.].

7.5. *Frontend*

7.5.1. *Web3*

Lo primero que se realizó fue crear un entorno de ReactJS e instalar las librerías necesarias para poder realizar la conexión entre *frontend* y el *smart contract* desarrollado.Una de ellas es la librería de *Web3* que es la encargada de traducir las llamadas a funciones del *smart contract* a llamadas por medio de transacciones y poder interactuar con el mismo por medio de la interacción de un nodo de Ethereum que fue el que se creo anteriormente cuando se desplegó el *smart contract*. También, se añadieron librerías como *react-router-dom* para manejar las rutas dentro del sitio y ChakraUI como librería de componentes y estilos para realizar el desarrollo de *frontend* de manera más ágil.

Una vez se instalaron todas las dependencias necesarias, se administró la conexión con un *injected provider*, en este caso la billetera de Metamask para poder firmar las transacciones que se realizan al *smart contract*. Esta conexión se realizó por medio del estándar EIP-1102 por medio de la librería de *Web3*. Se implementó funciones para que esta conexión sea sensible a cambios de red, así como a cambios de cuentas para que si un usuario se conecta con una cuenta valida, y luego cambia de red, a una que no es soportada por la plataforma, el usuario tenga retroalimentación de que no puede utilizar su billetera en esa red y únicamente puede utilizar las redes habilitadas que en este caso es la red principal de Ethereum , la red de prueba de Görli y la red de prueba Fuji de Avalanche. Luego, se creó el archivo de la página principal de la plataforma en donde se implementó un botón que ejecuta la conexión con Metamask, y una vez se realice la conexión muestre la dirección de la billetera, así como su saldo disponible.

Luego, se configuró un *Artifact* para poder instanciar el *smart contract* y poder interactuar con el, en donde se le indica a *Web3* la dirección en donde se encuentra el *smart contract* desplegado y la *jsonInterface* o ABI para que se puedan construir los JSON-RPC. De esta manera *Web3* construye los métodos, realizados en el *smart contract*, en Javascript. Posteriormente, se realizó un *Hook* para poder utilizar el contrato dentro de cualquier archivo de código.

Se implementó el *hook* realizado, en la pagina principal para poder habilitar la función de *mint*

y crear un *NFT* desde la plataforma *web*. Para implementarlo correctamente, primero se valida que el usuario este conectado por medio de Metamask para poder habilitar el botón de *mint*, de lo contrario un usuario no puede hacer clic ya que se utiliza Metamask para poder firmar las transacciones.

```
import { useMemo } from 'react';
import { useWeb3React } from '@web3-react/core';
import VestaArtifact from '../../config/web3/artifacts/Vesta';

const { address, abi } = VestaArtifact;

const useVesta = () => {
    const { active, library, chainId } = useWeb3React();

    const vesta = useMemo(() => {
        if (active) return new library.eth.Contract(abi, address[chainId])
    }, [active, chainId, library?.eth?.Contract])

    return vesta;
}

export default useVesta;
```

Script 7.20: (ReactJS) *Hook* para acceder a las funciones desarrolladas en el *smart contract*

Luego de esta validación, se desarrolló una función que primero llama a la función de *mint* del *smart contract* y luego ejecuta tres eventos para escuchar el proceso completo que se realiza en una transacción de *blockchain* y poder mostrar retroalimentación del usuarios. Los eventos que se escuchan son: *transaction hash* para mostrarle al usuario el numero de transacción cuando la transacción es confirmada y si está en ejecución por si desea consultarla en un explorador de bloques, *receipt* para anunciar que la transacción fue exitosa y ha finalizado, y error para evidenciar si sucede algún error en la transacción.

```
vesta.methods.mint().send({
    from: account,
    // value: 1e18,
}).on('transactionHash', txHash => {
    toast({
        title: 'Transaccion enviada',
        description: txHash,
        status: 'info'
    })
}).on('receipt', () => {
    setIsMinting(false);
    toast({
        title: 'Transaccion confirmada',
        description: 'La transaccion se ha concluido con exito',
        status: 'success'
    })
}).on('error', error => {
    setIsMinting(false);
    toast({
        title: 'Transaccion fallida',
        description: error.message,
        status: 'error'
    })
})
```

Script 7.21: (ReactJS) Implementación de *mint* en la plataforma *web*

Con el *mint* habilitado, se prosiguió en la creación de la pantalla de *marketplace* en donde se listan todos los *NFTs* que han sido creados. Para esto, primero se crearon ciertos componentes de interfaz como tarjetas y *spinners* para poder implementar de una manera agradable la vista de cada

NFT. Luego, se creó un componente en donde se listan. Luego se creó un *hook* personalizado en donde se obtiene los datos de los *NFT* (Ver anexo C.7). El *hook* consta de dos funciones, una que obtiene los datos de manera conjunta de todos los *NFT*, y otra que obtiene los datos de un *NFT* en especifico. Estas funciones utilizan una función que se encarga de ejecutar el método de *tokenURI* del *smart contract* para obtener los metadatos y también el método de *ownerOf* para obtener el dueño de un *NFT*. También, contiene una función encargada de traducir el resultado del *tokenURI* a JSON ya que el valor de retorno del método es una URL en base 64 y es necesario pasarla a JSON para poder leer cada uno de los datos almacenados en los metadatos. Para el *hook* de listar los *NFT* de manera conjunta, se realiza una llamada al método de *totalSupply* para conocer la cantidad de *NFTs* que se han creado para crear una estructura de dato de arreglo y poder iterar sobre este para que, por medio de una promesa, se obtengan todos los *NFTs* creados. Luego, se utilizó este *hook* en el componente creado para poder mostrarlos en el sitio *web*. Posteriormente, para el *hook* que obtiene los datos de un *NFT* en específico, se obtiene la información de la misma manera que en el *hook* que obtiene información de manera conjunta, la única diferencia es que no se realiza una consulta al *totalSupply* sino que se obtiene el token *id* del *NFT* y se realiza la consulta de datos con este *id*.

Con la vista de detalle de un *NFT* habilitada se procedió a la implementación de transferencia de *NFTs* entre billeteras para poder transferir la propiedad del mismo. Para esto se creó una función que primero valida que la dirección ingresada por el usuario a la cual se le realizará la transferencia del *NFT*, sea una dirección valida por medio de la utilidad de *isAdress* de *Web3*. Luego, si es una dirección valida, se ejecuta el método de *safeTransferFrom* del *smart contract* y se escuchan tres eventos, *transactionHash*, *receipt* y error, que cumplen la misma función como la implementación de la función *mint*.

```
const transfer = () => {
    setTransfering(true);

    const address = prompt("Ingresa la direccion de destino: ")

    const isAddress = library.utils.isAddress(address);

    if (!isAddress) {
        toast({
            title: 'Direccion invalida',
            description: 'La direccion no es una direccion de Ethereum',
            status: 'error'
        });

        setTransfering(false);
    } else {
        vesta.methods.safeTransferFrom(singleVesta.owner, address, singleVesta.
    tokenId).send({
            from: account
        }).on('transactionHash', (txHash) => {
            toast({
                title: 'Transaccion enviada',
                description: txHash,
                status: 'info',
            })
        }).on('receipt', () => {
            setTransfering(false);
            toast({
                title: 'Transaccion confirmada',
                description: `Su titulo de propiedad ha sido transferido con exito a
    ${address}`,
                status: 'success'
            })
            update()
        }).on('error', () => {
            setTransfering(false);
        });
```

```
36      }
37
38  }
```

Script 7.22: (ReactJS) Función para transferir la propiedad de un *NFT*

Por último, se filtraron los *NFTs* por dueño para que un usuario pueda acceder a sus *NFTs* desde su perfil. Esto se cumplió de la misma manera como se listaron los *NFTs* en la pantalla de *marketplace*, únicamente filtrando por dueño de *NFT* por medio de la utilización del método de *balanceOf* para conocer la cantidad de *NFTs* que posee un usuario y *tokenOfOwnerByIndex* del *smart contract* que fue implementado por medio del estándar del *ERC721Enumerable* para obtener el *NFT* según su *id*.

7.5.2. Integración *backend* con *blockchain*

La implementación de la integración entre ambas tecnologías es simple. La base de datos principal es la red de *blockchain*, esta red almacena los registros de las propiedades tokenizadas, información relevante de esta y su propietario, que es el dueño del *NFT*. El *backend* debe almacenar la información de los usuarios y las *wallets* de las que son dueños; de este modo se puede incluso guardar más información del usuario como una foto de perfil. Adicionalmente el *backend* debe almacenar en base de datos el identificador del *NFT* como parte de los datos de una Propiedad. De este modo se puede hacer referencia y buscar la propiedad por este token *id*.

Como se puede notar a nivel de base de datos solo se creó una tabla adicional a la tradicional *User* para almacenar las *wallets* que este posee y también se agrega otro identificador único a las tablas de las entidades que también almacena la red de *blockchain*. Siguiendo esta metodología se puede integrar sistemas *backend* con *blockchain*.

A nivel de la infraestructura de la nube del servidor no fue necesario hacer alguna implementación *custom* para soportar la integración. Al igual que al nivel del API. Donde se efectúan la mayor cantidad de cambios para poder soportar este tipo de integración es a nivel del *frontend*. En el *frontend* se implementó lógica que se encarga de utilizar *Web3* para hacer consultas a la red e inmediatamente después se debía hacer consultas al *backend* para obtener información adicional; entre esta información adicional se incluye validaciones de autenticación y autorización y proveer la información adicional que se solicita. El *backend* a su vez tiene cambios menores, como poder soportar búsquedas por el identificador de la red de *blockchain*. Adicionalmente, se puede creó de un oráculo que consulta y brinda información a la red de los precios de las propiedades [Ver Anexo E].

La integración se puede notar fuertemente en el *login* de la plataforma. Cuando queremos acceder y transaccionar en una red *blockchain* nos debemos conectar usando nuestra *wallet*. Actualmente hay servicios como Metamask que permiten hacer *login* a nuestras *wallets* y este nos proporciona la *wallet* o cuenta como tal para poder transaccionar. Sin embargo en la plataforma implementada luego de hacer login con Metamask y querer conectar la *wallet* a la plataforma para transaccionar, se hace una petición al *backend* consultando si la *wallet* que esta intentando conectarse ya esta registrada para algún usuario. Si se encuentra registrada el *backend* solicita un *password* para conceder un token de autenticación al usuario y que pueda navegar y transaccionar en la plataforma libremente. Si no se encuentra registrada la *wallet* en el *backend*, este solicita un registro que incluye información del usuario y se almacena. Y de este modo se logra eliminar el anonimato de la red y tener cierto control como en una estructura centralizada tradicional.

Hay otros puntos donde el *backend* se integra con el *blockchain*, como en el caso en el que un usuario este navegando entre todas las propiedades listadas. Este primer listado de propiedades proviene de los datos almacenados en *blockchain*, sin embargo cuando un usuario quiere acceder a ver mayor información de la propiedad se genera una consulta al *backend* para obtener el resto de datos de la propiedad como información adicional no almacenada en *blockchain* pero si almacenada en la base de datos e imágenes y archivos; los cuales no fueron almacenados en la red para reducción de costos transaccionales y de almacenamiento.

7.6. Infraestructura de nube

7.6.1. Tecnologías

Hay tres grandes del servicio de la infraestructura en la nube. Estos son Amazon Web Services, Google Cloud y Microsoft Azure. Los tres proveen múltiples servicios útiles y ecosistemas completos que pueden ser utilizados para montar infraestructuras en la nube. Cada uno tiene ventajas y desventajas en características, interfaz de usuario y precios. Para este proyecto se decidió que el aspecto principal por el que una infraestructura de nube es insegura, es por el error humano y el desconocimiento. Por eso se decidió enfocarse en soluciones que no provean tanta configuración y por esto mismo permitan más de estos errores humanos.

Se descubrió Amazon Lightsail, una herramienta que presenta Amazon Web Services como alternativa para los servicios tradicionales EC2, RDS y S3. Lightsail presenta una interfaz más simple que reduce las probabilidades de cometer errores al momento configurarlo. Por este motivo en particular se optó por utilizar Amazon Lightsail y no se consideró AWS, Google Cloud y Azure.

7.6.2. Implementación

La infraestructura de nube deseada consistiría de un servidor, base de datos y almacenamiento en la nube. Para lograr esto primero se accedió a Lightsail y nos dirigimos al apartado de instancias. Luego se procedió a crear una instancia, como se puede ver posteriormente solo se utilizó una instancia base con Ubuntu como sistema operativo. Adicionalmente a esta instancia se le asignó una IP estática y se habilitaron los puertos 22, 80 y 443. Finalmente se hizo un registro A en los DNS, para poder acceder a la IP estática asignada como subdominio del f-rosal.com; el subdominio que se le asignó fue vesta.f-rosal.com.

Después se creó una base de datos PostgreSQL en la nube. Crear una base de datos con Lightsail es un proceso sencillo que se puede visualizar posteriormente. Al igual que la base de datos, se creo un servicio de almacenamiento para archivos con Lightsail. Las imágenes de los procesos recién mencionados se listan a continuación [Ver Anexo G1-G3].

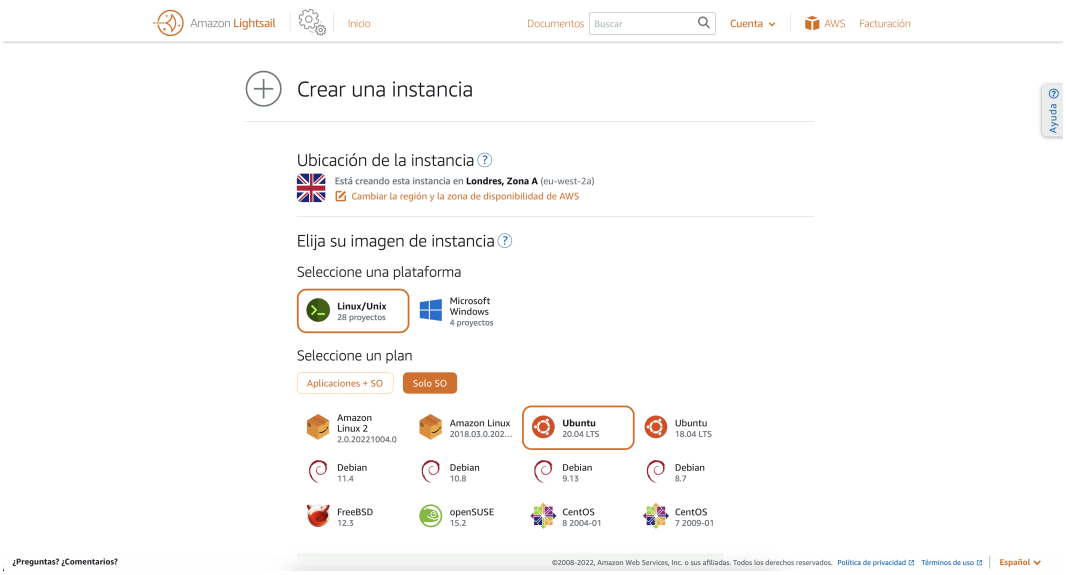


Figura 7.13: Creación instancia.

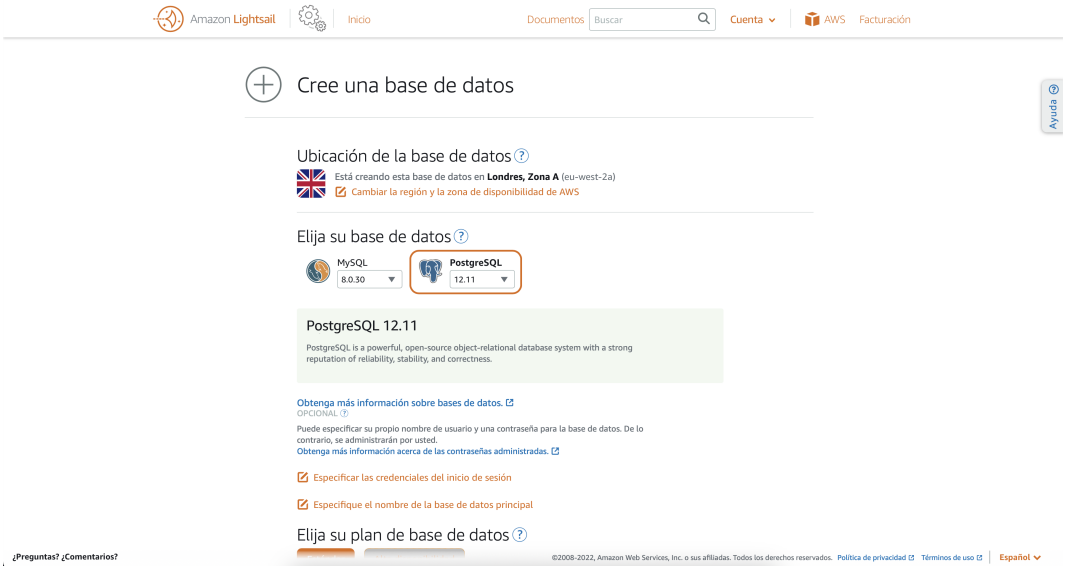


Figura 7.14: Creación base de datos.

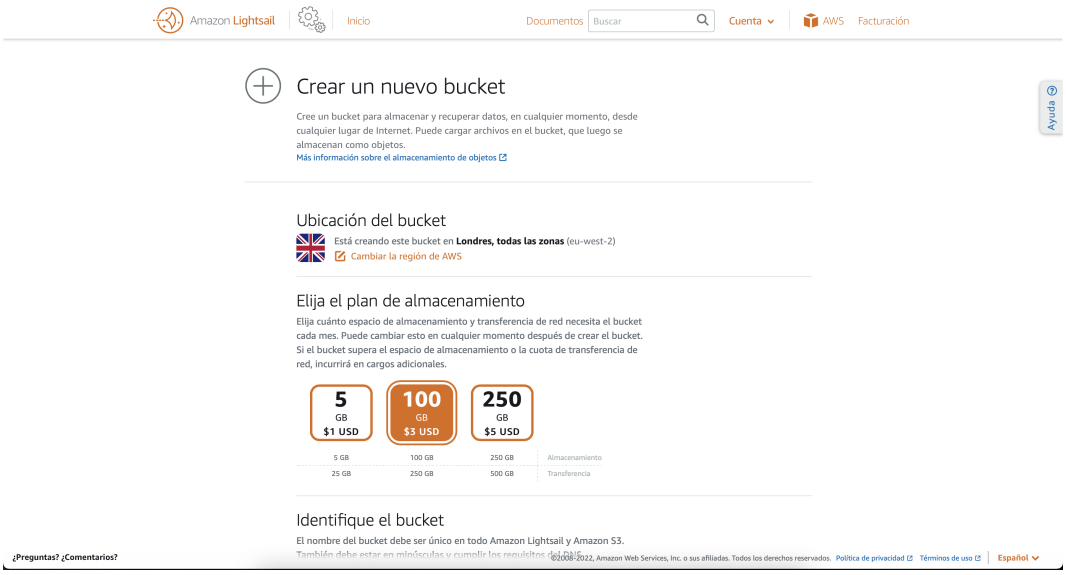


Figura 7.15: Creación almacenamiento.

Con toda la infraestructura del servidor en la nube montada correctamente se procedió a configurar el servidor, para esto se siguieron los pasos que se listan a continuación.

```
# Actualizacion del servidor
sudo apt update

# Instalacion de nginx
sudo apt-get install nginx

# Instalacion de dependencias de nginx
sudo apt-get install libnginx-mod-http-headers-more-filter

# Configuracion de certbot para obtencion del protocolo https - https://certbot.eff.org/instructions?ws=nginx&os=ubuntufocal
sudo snap install core; sudo snap refresh core

# Desinstalacion de cualquier instalacion previa de cerbot
sudo apt-get remove certbot

# Instalacion nueva de cerbot
sudo snap install --classic certbot

# Creacion de symbolic link
sudo ln -s /snap/bin/certbot /usr/bin/certbot

# Instalacion de Passenger (este nos ayuda en la comunicacion entre rails y nginx) https://www.phusionpassenger.com/docs/advanced_guides/install_and_upgrade/nginx/install/oss/focal.html
sudo apt-get install -y dirmngr gnupg apt-transport-https ca-certificates

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 561F9B9CAC40B2F7

sudo sh -c 'echo deb https://oss-binaries.phusionpassenger.com/apt/passenger focal main > /etc/apt/sources.list.d/passenger.list'

# Actualizacion del servidor
sudo apt-get update

# Instalacion de paquete passenger para nginx
```

```
sudo apt-get install -y libnginx-mod-http-passenger

# Configuracion de Passenger
if [ ! -f /etc/nginx/modules-enabled/50-mod-http-passenger.conf ]; then sudo ln -s /
    usr/share/nginx/modules-available/mod-http-passenger.load /etc/nginx/modules-
    enabled/50-mod-http-passenger.conf ; fi

sudo ls /etc/nginx/conf.d/mod-http-passenger.conf

# Verificar que el archivo /etc/nginx/conf.d/mod-http-passenger.conf exista

# Reinicio del servidor
sudo service nginx restart

# Validar configuracion conmla ejecuccion de
sudo /usr/bin/passenger-config validate-install
```

Script 7.23: (Bash) Instalación de Nginx y configuración de *Cerbot* y *Passenger*

```
# Nos dirigimos a la ubicacion de los sitios disponibles de ningx
cd /etc/nginx/sites-available

# Creamos el archivo virtual host
sudo nano vesta.f-rosal.com.conf
```

Script 7.24: (bash) Nginx *Virtual Host*

Se agrega el contenido al archivo.

```
server {
    # API

    server_name vesta.f-rosal.com;
    listen 3000;

    root /var/www/vesta.f-rosal.com/public;

    # Write log in packages instead of every event
    access_log /var/log/nginx/vesta.f-rosal.com.access.log combined buffer=16k;
    error_log  /var/log/nginx/vesta.f-rosal.com.error.log;

    client_max_body_size 10M;

    index index.html index.htm index.nginx-debian.html;

    passenger_enabled on;
    passenger_ruby /usr/share/rvm/gems/ruby-2.7.2/wrappers/ruby;

    # Disable header with server info
    server_tokens off;
    more_clear_headers Server;
    more_clear_headers 'X-Powered-By';

    # Block invalid requests (mostly from scanning tools)
    location ~ (\.jsp$|\.asp$|\.py$|\.perl$|\.php$|\.env$) {
        return 404;
    }

    # Rails assets
    location ~ ^/(assets)/  {

        root /var/www/vesta.f-rosal.com/public;

        # To serve pre-gzipped version
        gzip_static on;

```

```
        # Cache everything
        expires max;
        add_header Cache-Control public;

        # Do not log asset requests
        access_log off;
    }

    location / {

        # Always upgrade to HTTP/1.1
        proxy_http_version 1.1;

        # Enable keepalives
        proxy_set_header Connection "";

        # Optimize encoding
        proxy_set_header Accept-Encoding "";

    }

}

server {
    # SPA

    server_name vesta.f-rosal.com;

    listen 80;
    listen [::]:80;

    root /var/www/spa.vesta.f-rosal.com;

    # Write log in packages instead of every event
    access_log /var/log/nginx/spa.vesta.f-rosal.com.access.log combined buffer=16k;
    error_log  /var/log/nginx/spa.vesta.f-rosal.com.error.log;

    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

}

server {

    # It is always important to create a vhost to reject the direct access to the
    server throught IP address
    listen 80;
    server_name 18.169.147.97;
    return 404;

}

server {

    # It is always important to create a vhost to reject the direct access to the
    server throught IP address
    listen 3000;
    server_name 18.169.147.97;
    return 404;

}
```

Script 7.25: vesta.f-rosal.com.conf

```
# Instalacion de rvm para ubuntu https://github.com/rvm/ubuntu_rvm
```

```
sudo apt-get install software-properties-common
sudo apt-add-repository -y ppa:rael-gc/rvm
sudo apt-get update
sudo apt-get install rvm
sudo usermod -a -G rvm $USER
echo 'source "/etc/profile.d/rvm.sh"' >> ~/.bashrc

# Reiniciar servidor

# Instalacion de Ruby
rvm list known

rvm install 2.7.2

# Instalacion de dependencias de PostgreSQL
sudo apt-get -y install libpq-dev

# Generacion de llaves SSH para acceso a Github
cd /home/ubuntu/.ssh/

ssh-keygen

# Registrar llave ssh generada en Github copiando el contenido de id_rsa.pub
cat id_rsa.pub

cd /var/

# Hacer a ubuntu propietario del directorio www
sudo chown ubuntu www -R

cd www/

# Instalacion del codigo
git clone git@github.com:UVG-Teams/realestate-nft-marketplace.git

# Renombrar directorio
mv realestate-nft-marketplace/ vesta.f-rosal.com/

cd vesta.f-rosal.com

# Instalacion de gemas
BUNDLER_WITHOUT="development:test" bundle install

# Utilizacion de rama production
git checkout production

# Configuracion de credenciales (db, aws y otros)
EDITOR="nano" rails credentials:edit --environment production

# Creacion de DB y ejecucion de migraciones y seeders
RAILS_ENV=production rails db:drop

RAILS_ENV=production rails db:create

RAILS_ENV=production rails db:migrate

RAILS_ENV=production rails db:migrate:status

RAILS_ENV=production rails db:seed


# Creacion de symbolic links para habilitar el sitio
sudo ln -s /etc/nginx/sites-available/vesta.f-rosal.com.conf /etc/nginx/sites-enabled
    /

# Sitios activados
cd /etc/nginx/sites-enabled

```

```
# Revisar configuracion de Virtual Hosts
sudo nginx -t

# Reiniciar servidor
sudo service nginx restart

# Configuracion de certbot
sudo certbot --nginx

# Reiniciar servidor
sudo service nginx restart

cd /var/www/vesta.f-rosal.com/

# Agregar permisos de escritura al directorio public
sudo chmod 755 public/ -R

# Hacer a ubuntu propietario del directorio public
sudo chown ubuntu public/ -R

# Hacer a nginx propietario del directorio public
sudo chown www-data:www-data public/ -R

# Reiniciar servidor
sudo service nginx restart
```

Script 7.26: (Bash) Instalación y configuración del *backend* (API)

```
# Compilacion de archivos
npm run build

# Instalacion de paquetes necesarios https://github.com/Olafaloofian/React-Frontend-Lightsail-Deployment
sudo curl -sL https://deb.nodesource.com/setup_10.x -o nodesource_setup.sh

sudo bash nodesource_setup.sh

sudo apt-get install nodejs

sudo apt-get install build-essential

cd /var/www/

# Creacion de directorio para frontend
mkdir spa.vesta.f-rosal.com

# Creacion de symbolic link para archivos compilados del frontend
sudo ln -s /var/www/vesta.f-rosal.com/lib/spa/build/* /var/www/spa.vesta.f-rosal.com

# Reiniciar servidor
sudo service nginx restart

# Instalacion de serve https://create-react-app.dev/docs/deployment
sudo npm install -g serve

# Ejecucion de la aplicacion
serve -s build -l 80

# Reiniciar servidor
sudo service nginx restart
```

Script 7.27: (Bash) Instalación y configuración del *frontend*

7.7. Modelo de reconocimiento de imágenes

7.7.1. Obtención y preprocesamiento de datos

Para todos los entrenamientos y pruebas se usó el *dataset* de uso público ***House Room Image Dataset*** que puede ser encontrado en este enlace:

`https://www.kaggle.com/datasets/robinreni/house-rooms-image-dataset`

En éste se encuentran 5250 imágenes a color en formato JPG con una resolución de 224x224 px correspondientes a 5 ambientes de una casa promedio:

1. Sala
2. Cocina
3. Comedor
4. Dormitorio
5. Sanitario

Esos fueron los ambientes que se usaron para el entrenamiento de los modelos de reconocimiento.

Existen distintas maneras de construir modelos de inteligencia artificial que puedan resolver el problema de reconocer si una foto pertenece (o no) a un ambiente principal de una casa que se desea vender/comprar. Sin embargo, se encontró que el modelo de red neuronal convolucional es el que da mejores resultados en tareas de clasificación de imágenes [12]. Con esto en mente, aún era necesario definir si se implementaría un modelo de una red neuronal cuya salida (*output*) fuera una lista de probabilidades correspondientes a cada posible ambiente (modelo multi clase), en el cual un resultado con probabilidad baja para todas las clases será interpretado como un rechazo (no ambiente) y un resultado con un ambiente con alta probabilidad sería tratado como una aceptación, o si se desarrollarían una serie de modelos binarios en la que, cada uno de manera independiente, retornaran una probabilidad de que cada imagen perteneciera a cada ambiente. En este caso se tendrían que determinar umbrales de aceptación para cada ambiente, y solo se tomará como rechazo el caso en que todos los modelos den un resultado menor a su umbral respectivo. Ambos modelos fueron implementados para comparar efectividad de predicción y escoger el mejor.

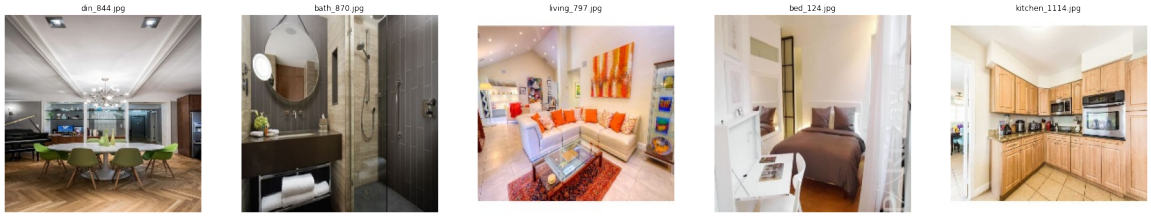


Figura 7.16: Imágenes aleatorias encontradas en el *dataset*.

7.7.2. Librerías y *hardware* utilizado

Para realizar el proceso de desarrollo, entrenamiento y pruebas se utilizó una instancia de Google Collab con las siguientes características:

1. Procesador Intel Xeon(R) CPU @ 2.30GHz
2. Tarjeta gráfica NVIDIA Tesla T4
3. 12.68 GB de memoria RAM disponible

Por otro lado, se usaron las siguientes librerías de Python 3:

1. numpy
2. pandas
3. matplotlib
4. cv2
5. seaborn
6. tensorflow
7. keras
8. matplotlib
9. random
10. os

Cabe mencionar que no fue posible usar la versión más reciente de la librería Tensorflow (2.10.0 al momento de redactar este documento) debido a un error interno de la librería a partir de la versión 2.9.0 (`https://github.com/tensorflow/tensorflow/issues/56242`) que causa que la velocidad del proceso de entrenamiento y pruebas de los modelos fuera inaceptable. Por lo que fue necesario regresar a la versión 2.8.0. Todas las demás librerías se usaron en su versión más actualizada.

7.7.3. Capas de preprocesamiento

Para reducir el tiempo de procesamiento requerido, y aumentar la potencial efectividad de cada modelo, todas las imágenes son cargadas en escala de grises, y reducidas a un tamaño de 224 x 224 pixeles.

El tamaño del *dataset* hacia que un modelo fuera propenso a sobre ajustarse a los datos de entrenamiento (hacer *overfitting*), de manera que no sería capaz de generalizar apropiadamente las características esenciales de cada ambiente y, por consiguiente, fallaría al intentar evaluar un conjunto de datos que no se encontrara en el conjunto de entrenamiento. Una de las mejores maneras de mitigar este problema es a través de la aplicación de una *augmentation layer* a las imágenes que servirán para el entrenamiento del modelo [95].

Para esto se desarrolló un modelo de preprocesamiento de datos en forma de red neuronal no entrenable que se encargó de, mientras se estuviera en un ambiente de entrenamiento, aplicar una serie de transformaciones aleatorias a las imágenes dentro de un rango especificado. De esta manera se evita que el modelo reciba la misma imagen varias veces, haciendo más difícil que se sobre ajuste a los datos de entrenamiento, forzándole entonces a enfocarse en aprender patrones de interés para

el problema.

Las transformaciones fueron las siguientes:

- Reflexión horizontal aleatoria
- Rotación aleatoria en un rango de -5 % a 5 % de una revolución completa.
- Ampliación horizontal en un rango de -5 % a 15 % (un valor negativo implica una reducción, o *zoom out*)
- Ampliación vertical en un rango de 1 % a 15 %
- Traslación horizontal en un rango de -5 % a 5 %
- Traslación vertical en un rango de -5 % a 5 %

En los casos en que fuera necesario, el método de llenado consideraba los píxeles más cercanos para calcular los datos faltantes, y el de interpolación por defecto fue bilineal. Los resultados de aplicar este modelo en la misma imagen varias veces de manera independiente se puede apreciar en la Figura 5.2.

7.7.4. Modelo de clasificación multiclase

Para este caso se implementó un modelo de red neuronal convolucional cuya primera capa era la de *augmentation* de datos previamente descrita, a la cual le seguían 3 parejas de capas convolucionales y de *max pooling*. La primera capa convolucional constaba de 5 filtros de 10x10, la segunda, con 10 filtros de 15 x 15, y la tercera con 20 filtros de 10 x 10. Las capas de *pooling* tenían filtros de 2x2 pixeles. A estas capas les segía una capa de aplanamiento (*flatten*), una capa densa completamente conectada de 150 neuronas y finalmente, la capa de salida, con 5 neuronas en total.//

Todas las capas que necesitaran usaron la función de activación ReLU, con excepción de la capa de salida, que usó *softmax* para garantizar que la salida del modelo fuera una distribución de probabilidad.

El conjunto de datos inicial se separó en 80 % para conjunto de entrenamiento y 20 % para el conjunto de pruebas. Se dejó entrenar por 150 iteraciones usando el optimizador `Adam`.

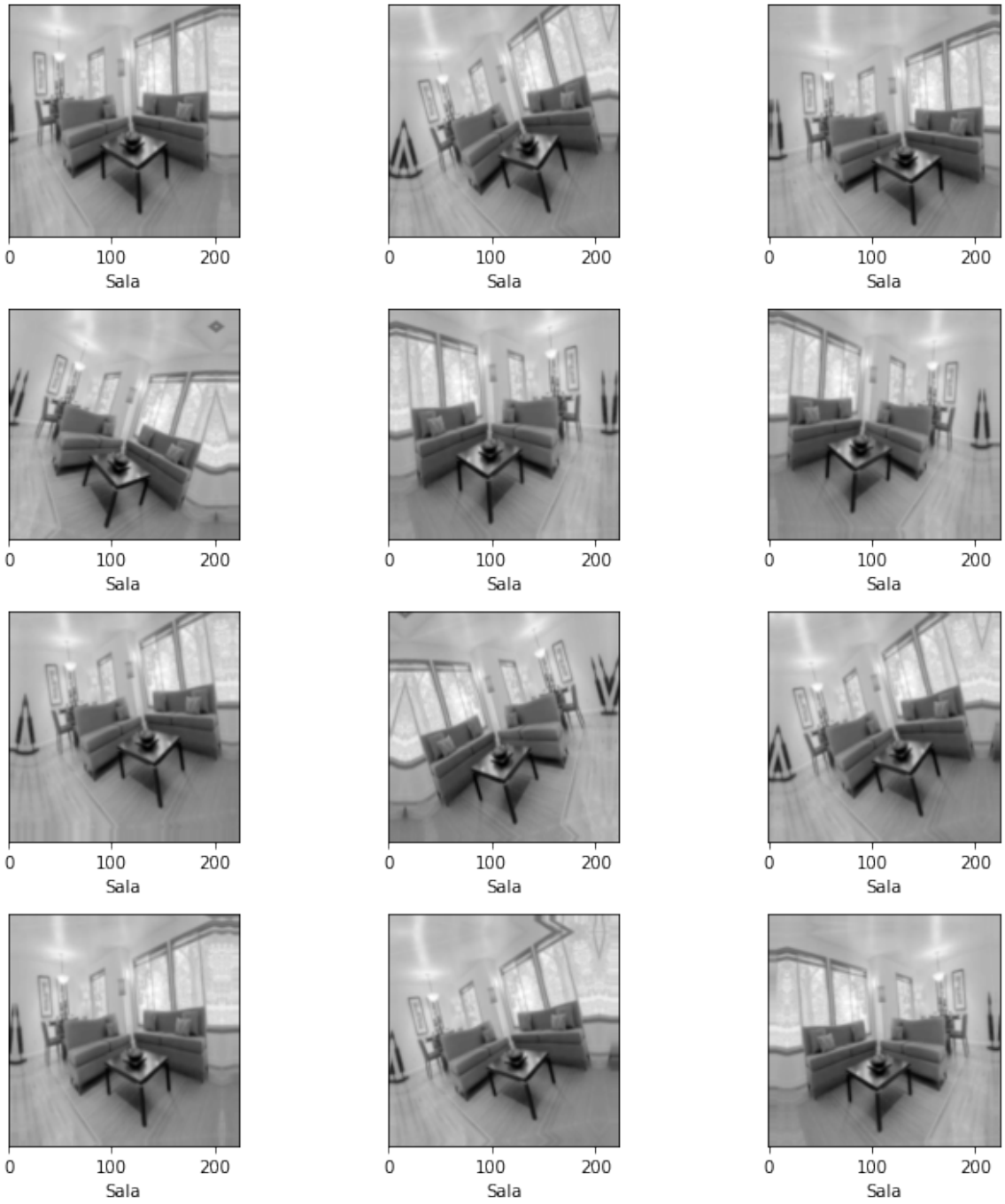


Figura 7.17: Una misma imagen al aplicar el modelo de *augmentation* distintas veces.

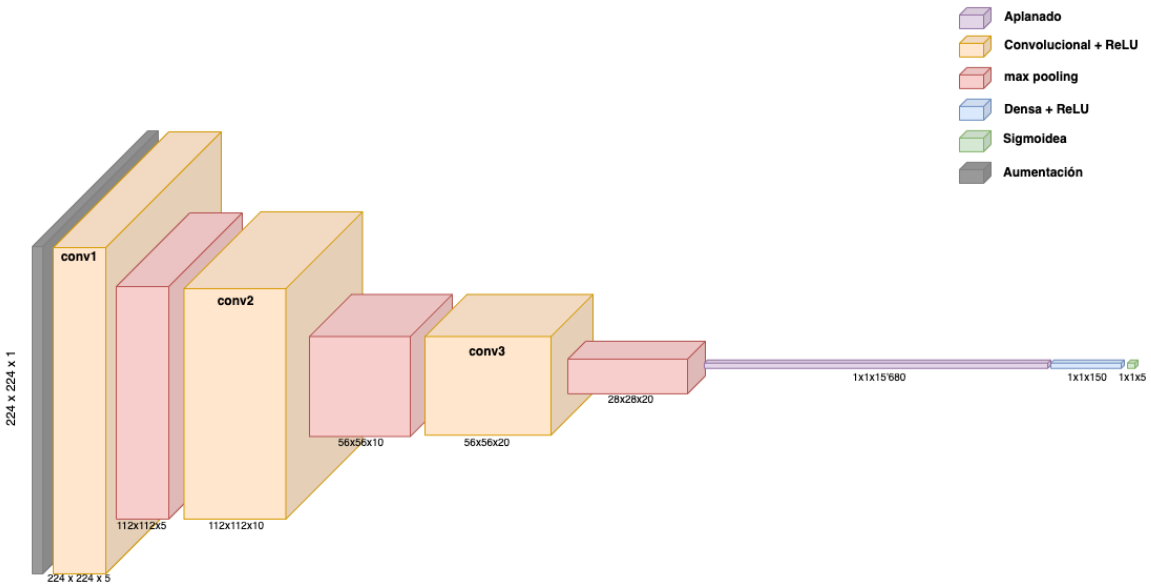


Figura 7.18: Arquitectura de red neuronal multiclase para clasificar imágenes en 5 grupos.

7.7.5. Modelos de clasificación binaria

Para este caso se implementó un modelo de clasificación por cada ambiente del *dataset* original (sanitario, cocina, sala, dormitorio y comedor). Cada modelo tiene como entrada una imagen sin clasificar y como salida dará un valor numérico en $[0, 1]$ representando la certeza del modelo de que la imagen corresponda a el ambiente que el modelo evalúa.

El conjunto de datos de entrenamiento fue obtenido del conjunto original usado para el modelo multiclase. Por cada ambiente que se deseaba clasificar, se generaron dos etiquetas: `yes_AMBIENTE` y `no_AMBIENTE`. En la primera etiqueta se encontraban todas las imágenes que sí deberían ser clasificadas como pertenecientes a `AMBIENTE`, mientras que en la segunda se encontraba una mezcla aleatoria de imágenes que se encontraban etiquetadas con las otras 4 etiquetas en el *dataset* original. La manera en la que se escogieron las imágenes que entrarían en esta segunda etiqueta fue a través de `random`, un módulo incluido en la distribución oficial de Python 3 que, con su función `randint`, se encarga de generar números pseudo aleatorios con distribución uniforme sobre el intervalo que se le establezca. [64] De esta manera, se pudo minimizar la posibilidad de sesgo entre los datos de entrenamiento del *dataset* pasado a cada modelo.

A todos los modelos se les colocó la *augmentation layer* usada en el modelo de clasificación multiclase, con la finalidad de evitar problemas de sobreajuste. Se dividieron los datos en un 80 % de entrenamiento y 20 % de pruebas.

Modelo de reconocimiento de sanitarios

Este modelo desarrollado consta de 10 capas, después de la *augmentation layer*:

- 3 parejas de Convolución/*Max Pooling*

- 1 capa de aplanado
- 1 capa densa de 150 neuronas
- 1 capa de tipo *dropout* que se encarga de regularizar el *dataset* en el entrenamiento para evitar problemas de sobre ajuste junto con la *augmentation layer*
- 1 capa final con la salida

La arquitectura se puede apreciar mejor en la Figura 7.19.

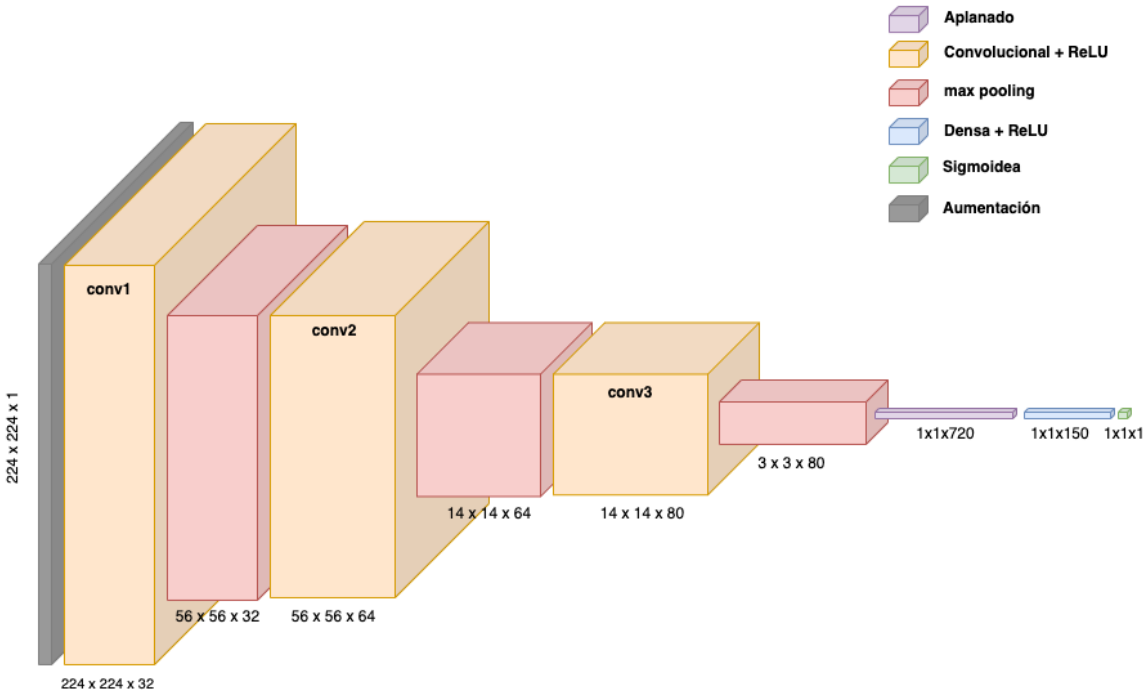


Figura 7.19: Arquitectura de la red neuronal de clasificación binaria de ambientes de sanitarios.

Modelo de reconocimiento de dormitorios

Este modelo usó la misma arquitectura que el modelo de reconocimiento de dormitorios, con excepción del mecanismo de *dropout*, que se aplica en el modelo de reconocimiento de dormitorios, pero no en este caso. Dado que el modelo no presentó efectos de sobreajustes en los entrenamientos iniciales, se decidió que no era necesario.

Modelo de reconocimiento de cocinas

Este modelo utilizó una arquitectura muy parecida a la usada en los modelos de sanitarios. Sin embargo, en éste las capas convolucionales usan una mayor cantidad de filtros (aunque se usan filtros del mismo tamaño que en el modelo anterior). Esto es de esperarse, pues es posible encontrar más elementos distintivos en una cocina que en una habitación. Como se puede notar en la Figura 7.20, cada capa de la red resulta ser más *ancha* porque las capas convolucionales evalúan más filtros.

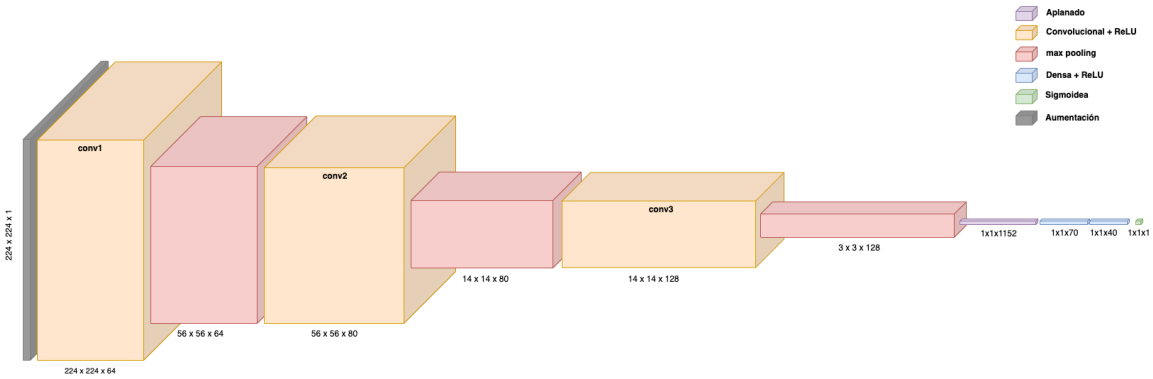


Figura 7.20: Arquitectura de la red neuronal de clasificación binaria de ambientes de cocinas.

Modelo de reconocimiento de comedores

Este modelo también aprovecha los buenos resultados que obtuvo la arquitectura del modelo de reconocimiento de sanitarios, con la diferencia que, ahora, se utilizan menos neuronas en la capa densa previa a la capa de salida (100 en este modelo, mientras que el modelo de sanitarios utiliza 150).

Modelo de reconocimiento de salas

Para desarrollar este modelo se tomó inspiración en en la arquitectura AlexNet[61] luego de agotar recursos haciendo pruebas con arquitecturas similares a la de reconocimiento de sanitarios y obtener resultados que no eran significativamente mejores que una clasificación aleatoria (50 % de efectividad). La arquitectura final para este modelo contó con:

- 5 capas convolucionales usando ReLU como función de activación
- 3 capas de *max pooing*
- 3 capas densas completamente conectadas de manera consecutiva, usando ReLU como función de activación.
- 1 capa de *dropout* ubicada entre la primera y segunda capa densa
- 1 capa de 1 neurona final con una función de activación sigmoide

Además, este fue el único modelo que requirió que se usara un optimizador de descenso estocástico de gradiente con una taza de aprendizaje explícita de 0.001. Como este valor fue tan pequeño, hubo necesidad de usar 400 rondas de entrenamiento, contrario a las 250 de los demás modelos.

En total el modelo tuvo 24,229,369 parámetros entrenables. Una descripción más detallada de la arquitectura se puede apreciar en la Figura 7.21.

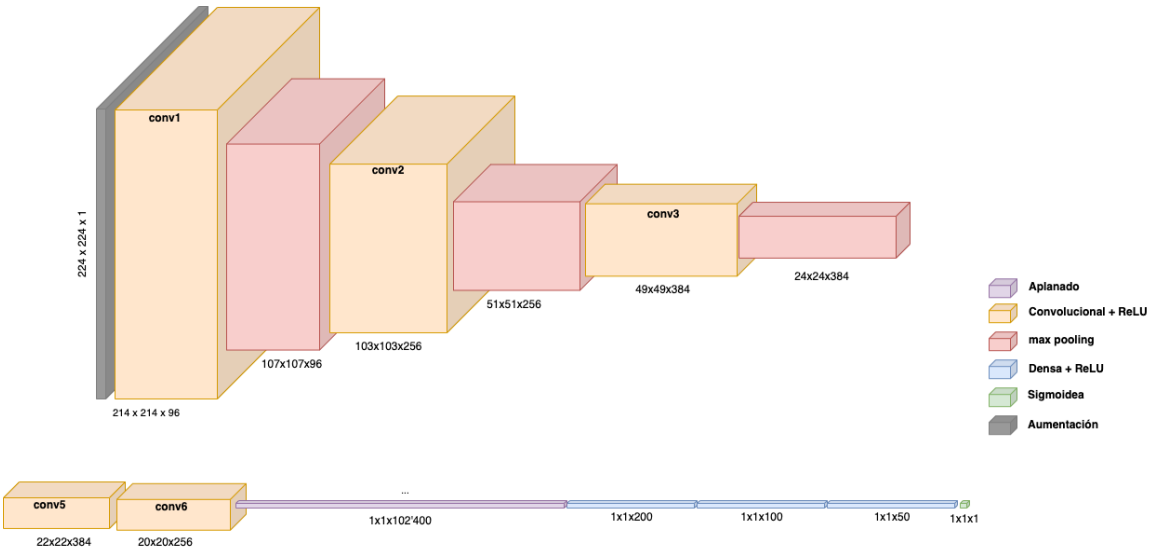


Figura 7.21: Arquitectura de la red neuronal de clasificación binaria de ambientes de sala.

7.8. Sistema de predicción de precios

7.8.1. Herramientas utilizadas

Para desarrollar el sistema de sugerencia de precios se utilizaron, principalmente, las siguientes librerías de Python 3 en un a instancia de Google Collab con GPU NVIDIA Tesla T4 y 25GB de memoria RAM:

- pandas
- seaborn
- matplotlib
- sklearn
- XGBoost
- numpy
- joblib

7.8.2. Obtención y exploración de datos

Se intentó usar información del censo de población y vivienda de Guatemala del 2018. Sin embargo, no fue posible acceder a la información de manera directa a través de archivos planos o conexiones a bases de datos. Por tanto, se tuvo la necesidad de buscar información en censos de otros países, y la base de datos más completa que se encontró fue la del censo de Estados Unidos de 2019.

La Oficina del Censo de Estados Unidos (*US Census Bureau*) se encarga de realizar estudios estadísticos a la población estadounidense con frecuencia, entre ellos se encuentran:

- *American Community Survey* (ACS)
- *American Housing Survey* (AHS)
- *Annual Business Survey* (ABS)
- *Annual Survey of Manufactures* (ASM)

Para este desarrollo se usaron los datos de la *American Housing Survey* (AHS).

Esta encuesta (AHS) está patrocinada por el Departamento de Vivienda y Desarrollo Urbano (HUD por sus siglas en inglés) y realizada por la Oficina del Censo. Ha sido la encuesta nacional de vivienda más completa en los Estados Unidos desde su inicio en 1973, brindando información actual sobre el tamaño, la composición y la calidad de la vivienda de la nación y midiendo los cambios en el *stock* de viviendas a medida que envejece. La AHS es una encuesta de unidad de vivienda longitudinal realizada cada dos años en años impares, con muestras recalculadas en 1985 y 2015 [55].

Estos datos sí se encuentran disponibles en formato de *Public Use File* (PUF), que contiene un conjunto de tablas en las cuales cada fila representa una unidad de vivienda relacionadas a través de un *ID* único en cada una, y un formato *PUF Flat File*, en el cual está la información de todas las tablas del PUF, pero condensadas en una sola usando el identificador. Al momento de realizar el proyecto, los datos más actualizados eran los de la encuesta de 2019, éstos se pueden encontrar para uso libre en el siguiente enlace:

`https://www.census.gov/programs-surveys/ahs/data/2019/ahs-2019-public-use-file--puf-/ahs-2019-national-public-use-file--puf-.html`

Exploración de datos

Una vez descargado el *Flat file* se procedió a comenzar con la exploración de data. En este proceso se encontró que existen 63 mil 185 registros en el *dataset*, y que cada uno cuenta con 543 columnas. Una de ellas, llamada **MARKETVAL** almacena el valor de esa unidad de vivienda en el mercado si estuviera a la venta en el momento que se realizó la encuesta. Una descripción breve del contenido de esa columna se encuentra en la Tabla 7.1. Como se puede observar, más del 75 % de los datos se encuentran debajo de $\$10^6$.

count	$3.839 * 10^4$
mean	$3.762 * 10^5$
std	$5.537 * 10^5$
min	$1.000 * 10^3$
25 %	$1.404 * 10^5$
50 %	$2.552 * 10^5$
75 %	$4.359 * 10^5$
max	$9.999 * 10^6$

Tabla 7.1: Descripción de los datos en la columna *MARKETVAL* de la *Flat File* del AHS 2019

A continuación se procedió a explorar y visualizar varias columnas dentro del *dataset* para ganar entendimiento de la distribución de datos con la que se contaba. Las figuras 7.22 hasta 7.31 muestran algunos de los hallazgos de la exploración. Para revisar todos los campos explorados en esta etapa

ver el repositorio del proyecto (enlace en anexo I).

Luego de haber explorado los datos y haber comprendido que transformaciones eran necesarias para poder trabajar con ella, se procedió a la limpiar el *dataset*. En esta parte se escogieron individualmente una lista de columnas sobre las cuales se aplicarán los algoritmos de regresión en el siguiente paso. El algoritmo completo se puede encontrar en el anexo C.10 y la lista de columnas finales en el anexo H.

Finalmente, para obtener el listado de columnas a ser utilizadas en el modelo de predicción final se usó una matriz de correlación en donde se cruzaron todas las variables *limpias* del conjunto de datos y se seleccionaron únicamente aquellas que tuvieran una correlación con en el precio mayor a 0.1.

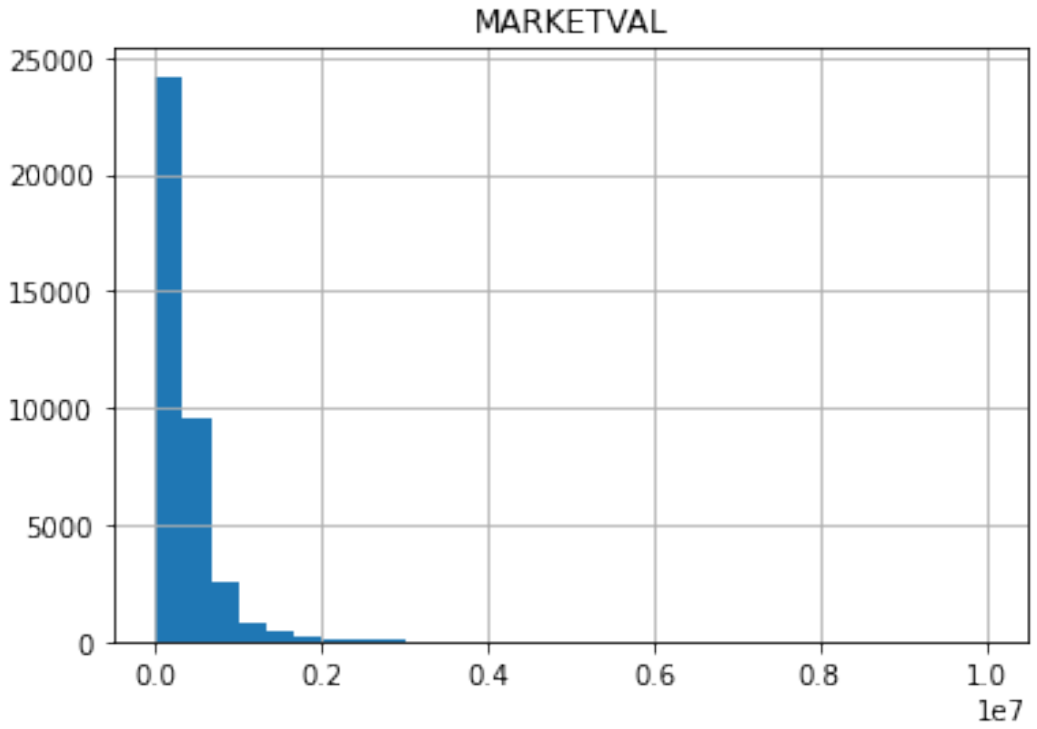


Figura 7.22: Histograma de precio actual en el mercado de las casas.

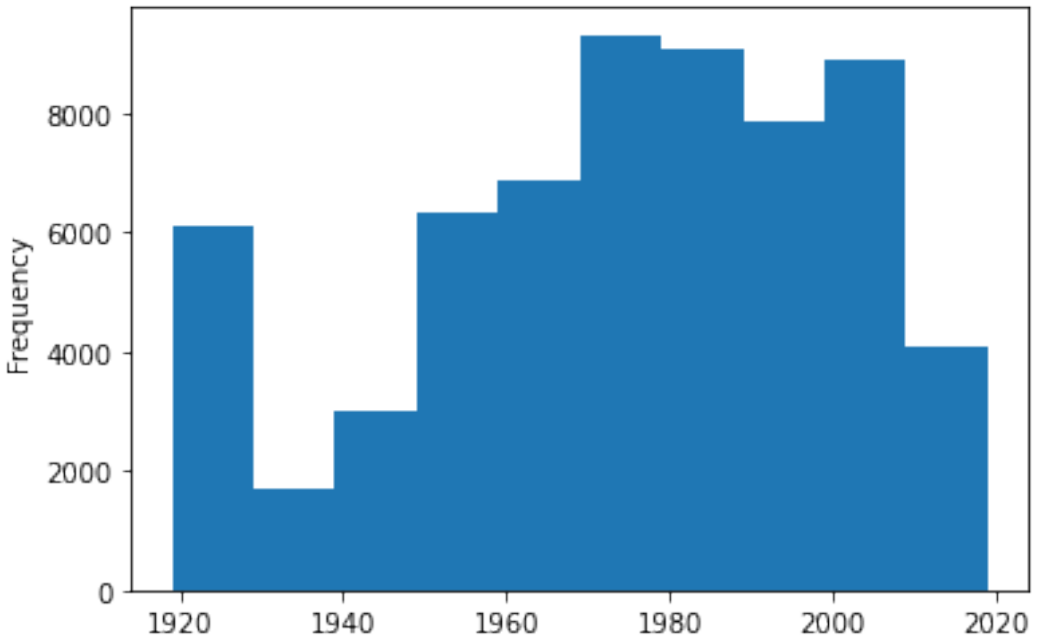


Figura 7.23: Histograma de año de construcción de casas.

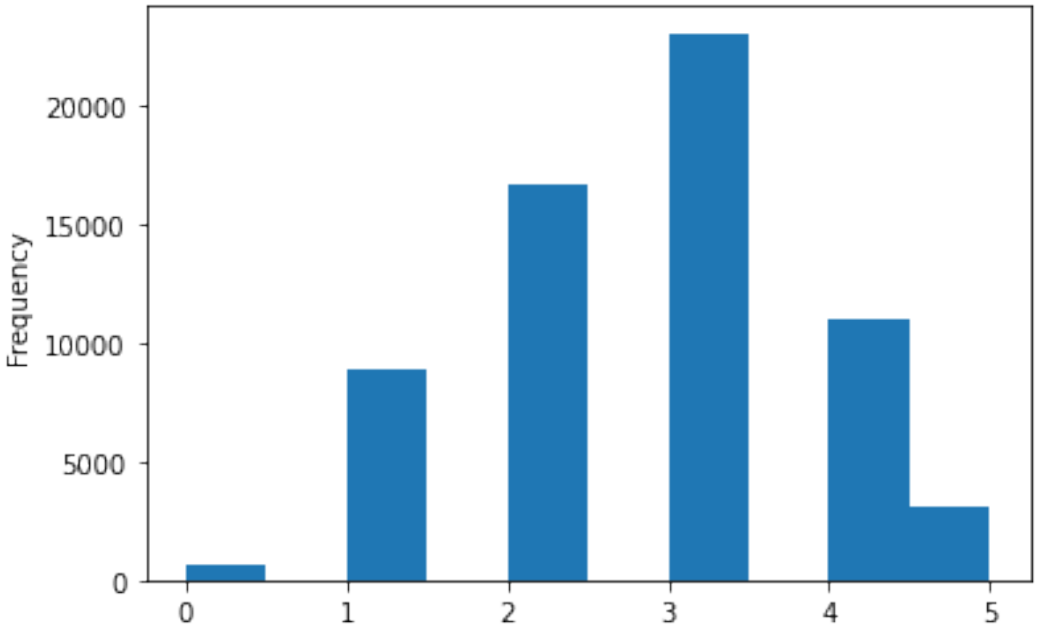


Figura 7.24: Histograma de cantidad de habitaciones en una casa.

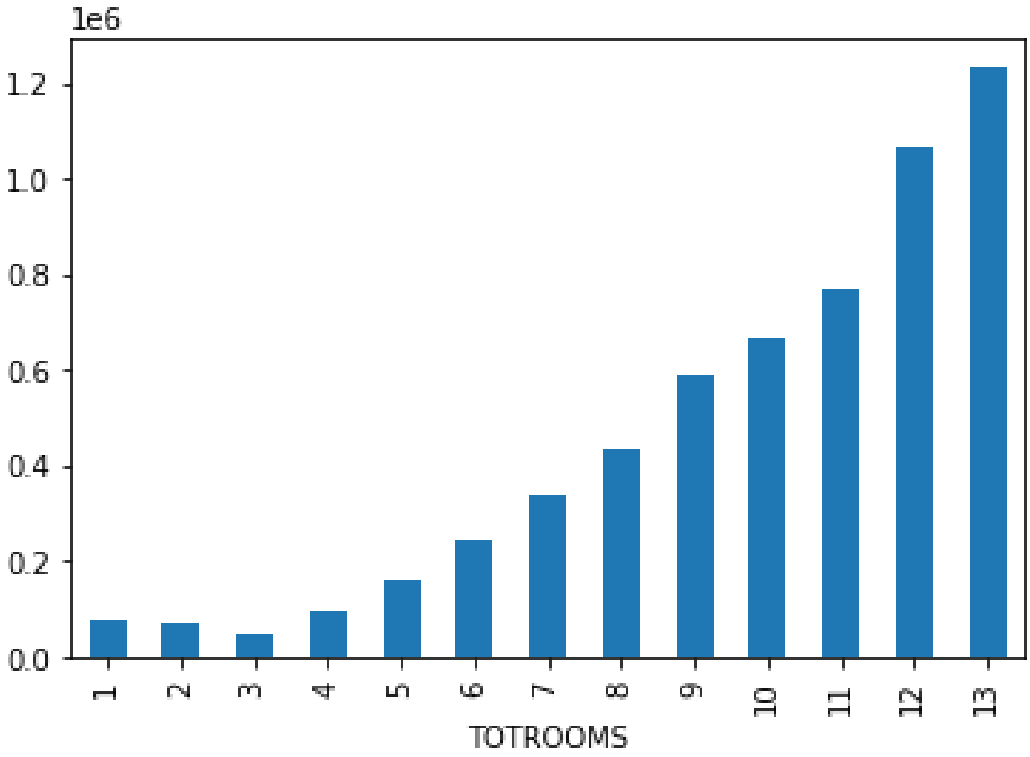


Figura 7.25: Media de precio de mercado por esas casas por cantidad de habitaciones.

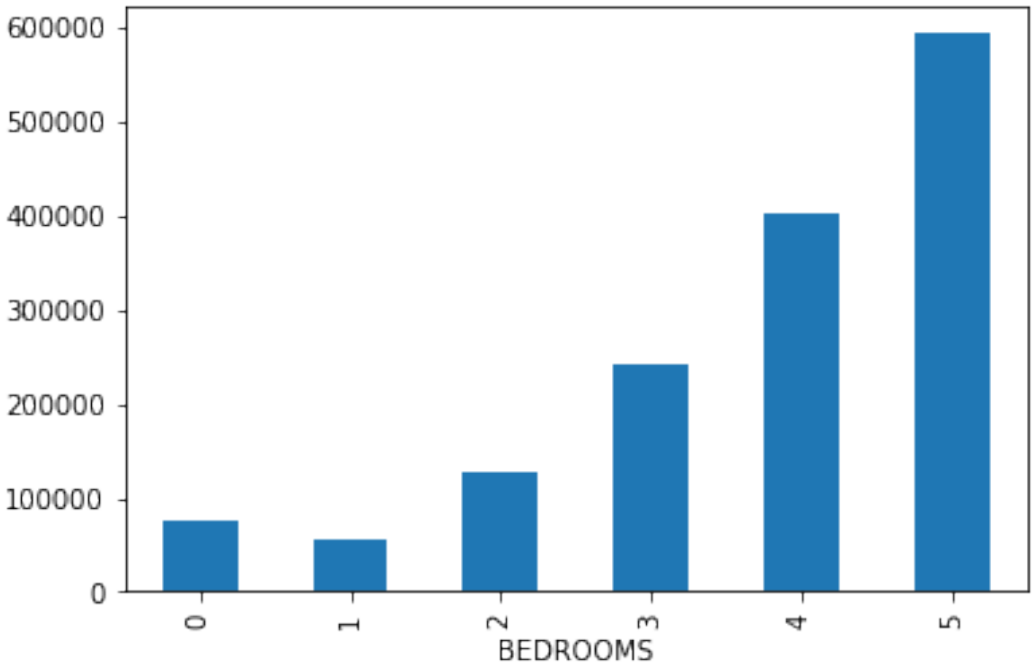


Figura 7.26: Media de precio de mercado por casas por cantidad de dormitorios.

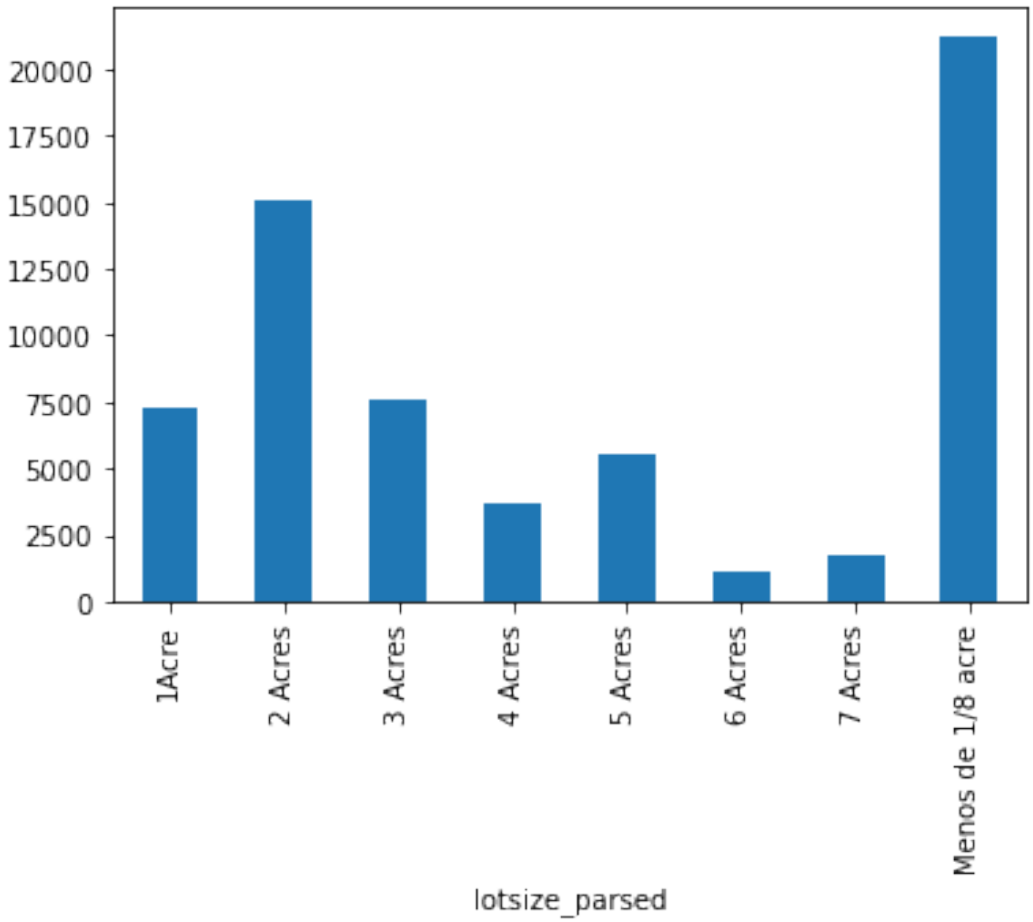


Figura 7.27: Cantidad de registros por tamaño de lote de propiedad.

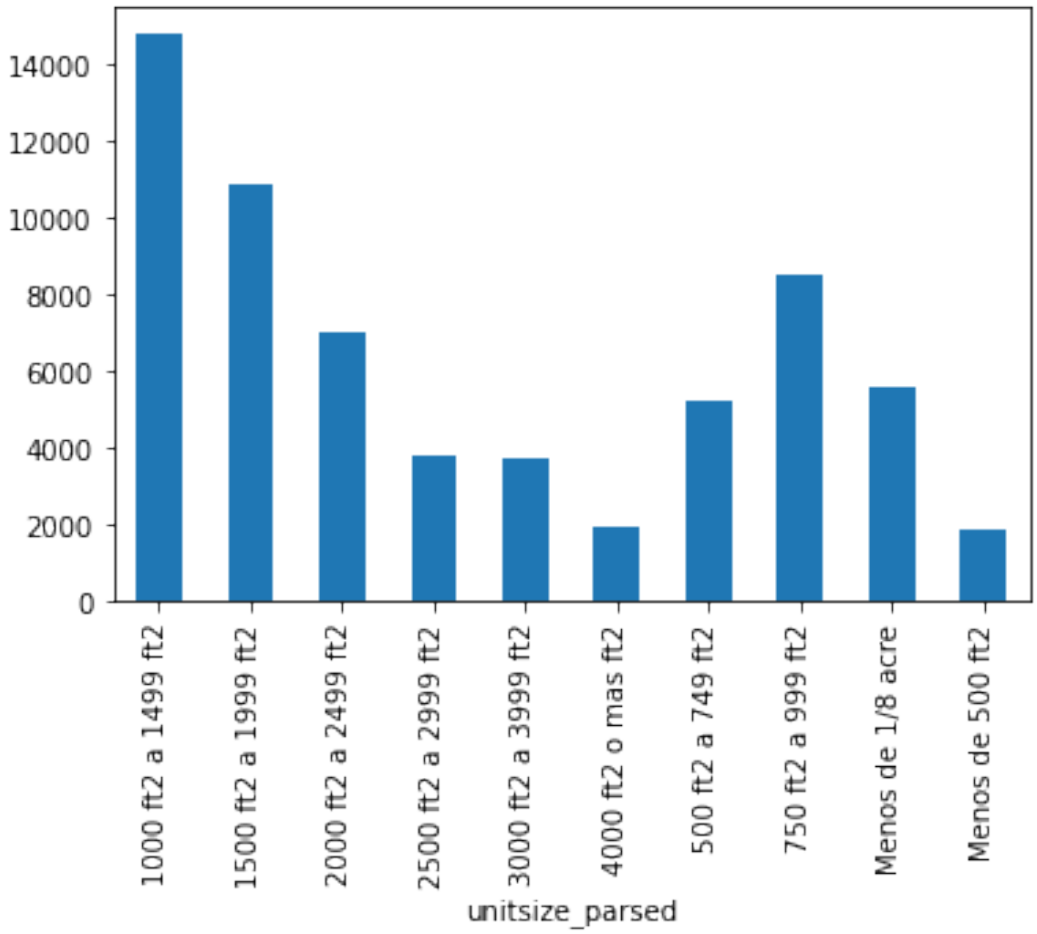


Figura 7.28: Cantidad de casas por tamaño de lote de construcción.

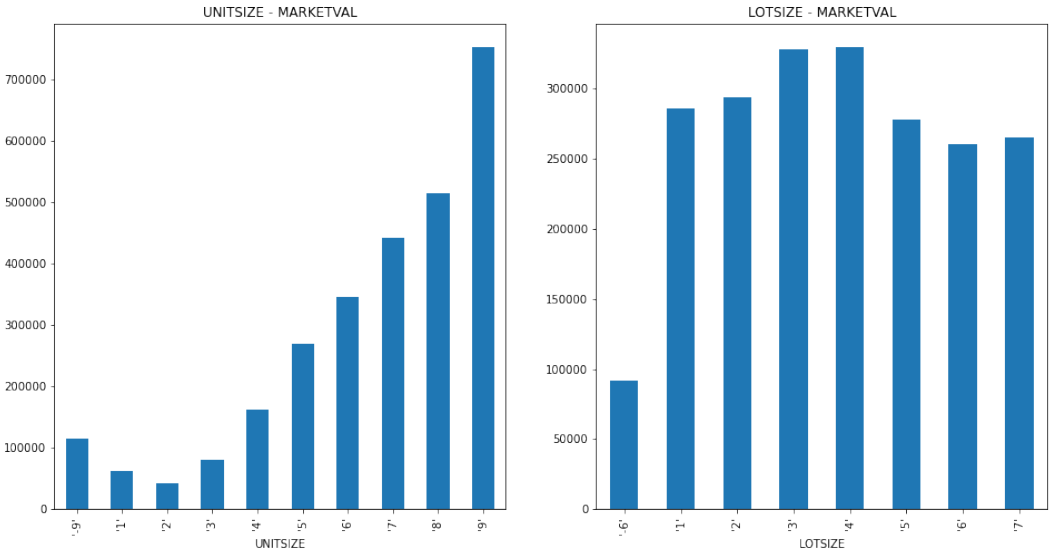


Figura 7.29: Precio en el mercado por tamaño de construcción y de lote.

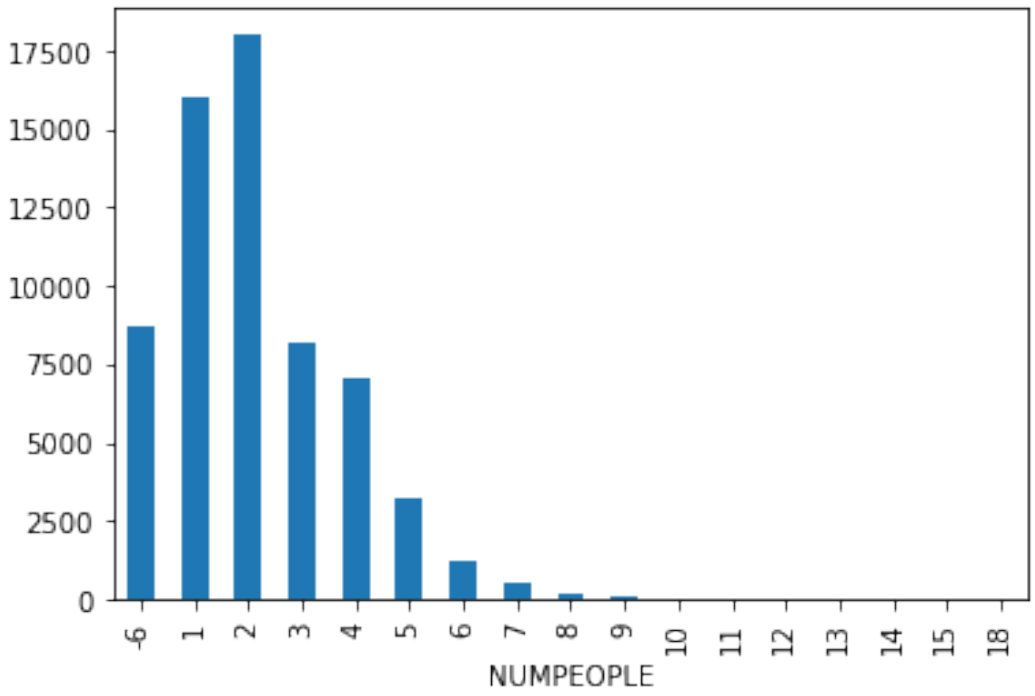


Figura 7.30: Histograma de cantidad de personas en una propiedad.

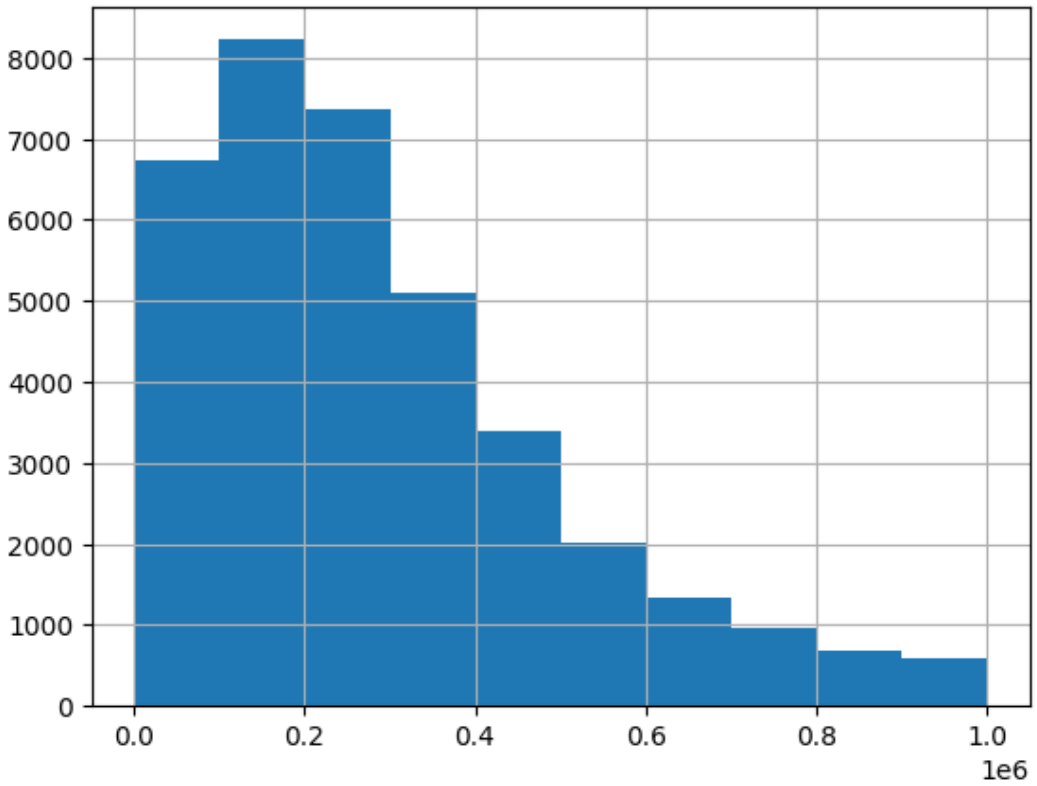


Figura 7.31: Histograma de precio en mercado - Únicamente considerando precios menores a 10^6.

7.9. API modelos de datos

7.9.1. Herramientas utilizadas

Para desarrollar el API web que usarán los sistemas de reconocimiento de imágenes y sugerencias de precios se decidió seguir lineamientos de la arquitectura REST. Principalmente, porque brinda flexibilidad para que se usen los mismos servicios en distintos lugares del resto del sistema, volviendo el código mas fácilmente reutilizable y eficiente, pero también porque seguir un patrón *Stateless* como el que sugiere REST[104] mejora la experiencia de desarrollo y reduce la complejidad final del módulo.

Para el desarrollo del servicio de *Backend* del sistema fue necesario decidir el lenguaje y *Framework* a utilizar. Dado que las demás herramientas ya habían sido desarrolladas usando Python, se decidió continuar el desarrollo usando el mismo lenguaje. Con respecto a el *framework* a usar, se tomaron en consideración algunos de los más populares actualmente en la industria:

1. Django

   Probablemente el más usado en la actualidad, es un *framework* gratis y de código abierto que se enfoca en facilitar el desarrollo de *apps web* de manera rápida y eficiente. Es usada popularmente para el desarrollo de *apps web* y APIs.

   Agiliza el desarrollo de aplicaciones web al brindar diferentes características vigorosas. Tiene una amplia variedad de bibliotecas compatibles y destaca la eficiencia, la menor necesidad de lineas de código y la reutilización de los componentes. [35]

2. CherryPy

   Tiene cerca de diez años de antigüedad, pero ha demostrado ser excepcionalmente rápido y estable. También es de código abierto e incorpora su propio servidor web WSGI. Puede ejecutarse en cualquier marco de trabajo que admita Python. [35]

3. Pyramid

   Pyramid es un *framework* flexible que permite a los desarrolladores programar aplicaciones web usando un enfoque minimalista. Se caracteriza por ser ágil, versátil y ser apto para proyectos tanto de pequeño o gran alcance. [35]

4. Flask

   Se trata de un *framework* accesible bajo la licencia BSD. Está inspirado en el marco Sinatra Ruby y se basa en la caja de herramientas Werkzeug WSGI y la plantilla Jinja2. El objetivo principal es ayudar a desarrollar una sólida base de aplicaciones web.

   Permite mucha flexibilidad en cuanto a diseño de aplicaciones. Sin embargo, fue diseñado para aplicaciones abiertas. Flask ha sido utilizado por grandes empresas, que incluyen LinkedIn y Pinterest. En comparación con Django, Flask es más adecuado para proyectos pequeños. Incluye manejo de solicitudes REST, *debugger*, servidor *web* http/https y es 100 % compatible con WSGI. [35]

Finalmente se decidió usar **Flask** por su compatibilidad nativa con WSGI y el patrón REST, su diseño orientado en proyectos puntuales y la familiaridad con el entorno de trabajo.

7.9.2. Consideraciones de seguridad

Una vez determinadas las herramientas con las cuales se realizaría el desarrollo, fue necesario tomar en consideración algunos potenciales problemas de seguridad que son comunes en APIs expuestas de manera pública en internet, en particular el riesgo de ataques de denegación de servicio (**DoS**).

En estos ataques, un atacante o grupo de atacantes pueden coordinar una cantidad de solicitudes HTTP que, de ser aceptada, excedan las capacidades de procesamiento del servidor que aloja el proceso del *backend*, de manera que aparezca como inalcanzable o inhabilitado para usuarios legítimos. [100] En el contexto del sistema de compraventa de bienes mobiliarios, este riesgo atenta directamente en contra del objetivo principal, que es el de optimizar y automatizar el proceso de compraventa.

Una buena manera de mitigar este riesgo es implementar un sistema robusto de autenticación que defina si una solicitud puede o no ser procesada. En este caso se decidió usar JSON *Web Token* (JWT). Se trata de un medio compacto y seguro para URL de representar solicitudes entre dos partes. Éstas solicitudes en un JWT están codificados como un objeto JSON que se utiliza como carga útil de un JSON *Web Signature* (JWS) o como texto plano en una JSON *Web Signature* (JWE), que permite que las solicitudes sean firmadas digitalmente o protegidas por integridad con un código de autenticación de mensajes (MAC) y/o cifrada. [58]

7.9.3. Modelos en base de datos

Tomando en cuenta que el sistema únicamente se encargará de analizar imágenes que reciba a través de una solicitud HTTPS (almacenarlas de manera independiente queda fuera del alcance del módulo), se decidió que la única información que se debía almacenar era aquella que fuera necesaria para autenticar a un usuario de manera única: usuario y contraseña. Por esta misma razón, no fue necesario tomar en consideración modelos no relacionales para el almacenamiento de información, y se implementó una base de datos relacional usando sqlite3.

Con la información almacenada es posible generar un token JWT que sirva para validar solicitudes en cualquier momento. El código para generar dicho token es el siguiente:

```
def authenticate(username, password):
    db = get_db()
    error = None
    if not username:
        error = 'Username is required.'
    elif not password:
        error = 'Password is required.'
    if error is None:
        user = db.execute(
            'SELECT * FROM user WHERE username = ?', (username,)
        ).fetchone()
        match = check_password_hash(user['password'], password)
        if user and match:
            return User(user['id'], user['username'])
    return {
        'message': error
    }
```

En donde el modelo `User` se define de la siguiente manera:

```python
class User(object):
    def __init__(self, id, username):
        self.id = id
        self.username = username

    def __str__(self):
        return f"User(id='{self.id}')"
```

7.9.4. Puntos de entrada

El API desarrollado expone 4 puntos de entrada (*endpoints*). Todos reciben información a través del cuerpo (*body*) de la solicitud y leen el contenido en formato *Form-data*:

1. POST /auth

   Recibe los datos de presunto usuario y contraseña y retorna un JWT token en caso la autenticación haya sido exitosa, o un mensaje de error en caso contrario.

2. POST /user/register

   Recibe el nuevo nombre de usuario y contraseña del usuario. Retorna un el nombre de usuario recién creado en caso la solicitud sea satisfactoria, o un mensaje de error en caso contrario.

3. POST /validate_image/

   Recibe el JWT token que autorice al usuario de hacer este tipo de solicitudes en el encabezado de la solicitud, y la imagen en el cuerpo. Retorna un valor de `True` si la imagen pertenece a uno de los ambientes evaluados, o `False` de lo contrario.

4. POST /predict_price/

   Recibe el JWT token que autorice al usuario de hacer este tipo de solicitudes en el encabezado de la solicitud, y los datos relevantes de la propiedad en el cuerpo. Retorna el valor estimado del precio de dicha propiedad, junto con un margen de error que se debe considerar.

7.10. Controles de acceso

Para la plataforma se decidió utilizar para la parte de identificación del usuario el correo electrónico, ya que de esta forma también nos facilita a comprobar si un usuario ya se había registrado, a comparación de usar nombre y apellido que hay probabilidades de que se repitan. Además, como nuestra plataforma utiliza *blockchain* para el traspaso de títulos también está la opción de poder registrarse con su billetera de criptomonedas a través de Metamask, que tiene *plugins* en varios navegadores para poder hacer la conexión con su cuenta. Adicionalmente se le solicita al usuario que ingrese se nombre y teléfono.

Por otro lado para la autenticación es importante que sea a través de algo que el usuario sabe, tiene o es. En este caso se usó la contraseña que es algo que el usuario sabe. Para que las contraseñas sean más seguras se limitó la cantidad de caracteres para la contraseña a seis como mínimo y esta debe contener al menos una letra minúscula, una mayúscula y un número, de esta forma ayudamos a que sea mucho más difícil que puedan obtener las contraseña de alguno de nuestros usuarios. La implementación del registro en el *backend* se puede ver en el anexo C.1 ahí está el método de creación del usuario en donde se busca si ya hay un usuario registrado, de ser así se regresa un error. Luego se verifica la complejidad de la contraseña y que la confirmación de la contraseña sea igual. Si todos

los datos son correctos se procede a crear al usuario. De igual manera en el anexo C.9 se puede ver la implementación en el *frontend* con las mismas verificaciones que en el *backend*. De momento no existen roles para los usuarios cualquier persona que se registre a la plataforma puede tanto ver las propiedades que hay en la plataforma para comprar, como publicar su propiedad para venderla.

7.11. KYC

Para complementar las otras formas de autenticación faltantes que es por medio de algo que tengo y soy se implementó KYC, en donde se autentica al usuario por medio de un documento y una foto de su rostro. Para ello, se hizo una investigación de los mejores *softwares* que proveen KYC y se hizo una comparación entre estos para determinar cual utilizar. Se comparó Faceki y Sentry, ambos proveen la autenticación que se necesita, sin embargo, Faceki nos ofrece mucha más variedad al momento de decidir cómo implementar el KYC, ya que se puede hacer por medio de un API, implementándolo en *web* o móvil y a través de un *link*. Además, que Sentry está más pensada para bancos. Al ya haber definido Faceki como el software a utilizar se realizó la configuración, primero se debe definir los países en los cuales los usuarios podrán iniciar sesión. Luego, se debe elegir los documentos que serán válidos para autenticar en nuestro caso seleccionamos el documento de identificación. Por último, se debe escribir las url's a las que se redirigirá cuando la autenticación sea inválida o válida. Para la implementación se decidió hacerlo mediante el *link* en donde se redirige al usuario para que suba los archivos necesarios para la autenticación.

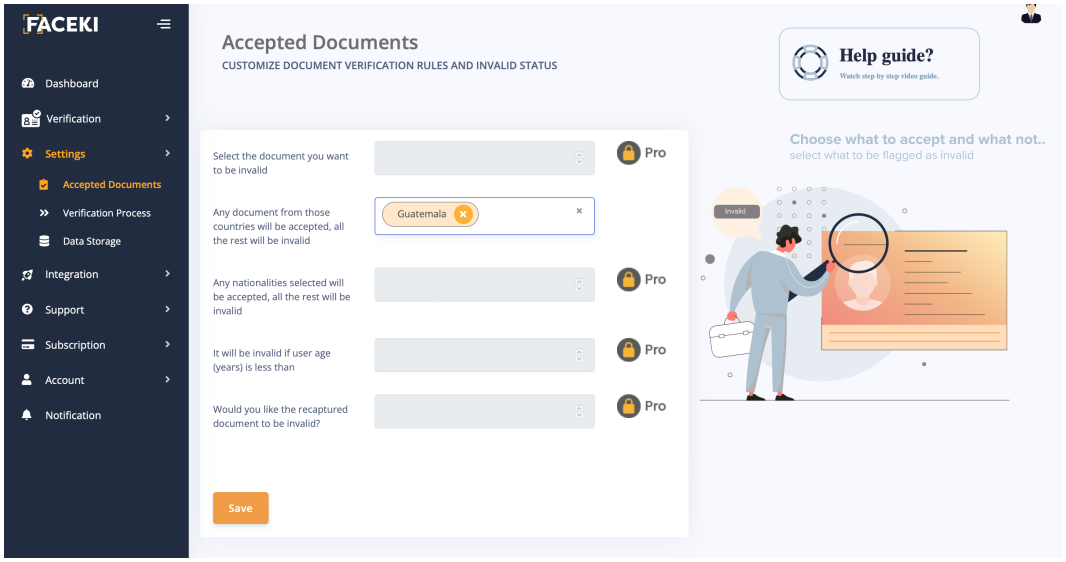


Figura 7.32: Configuración de país para autenticación con KYC

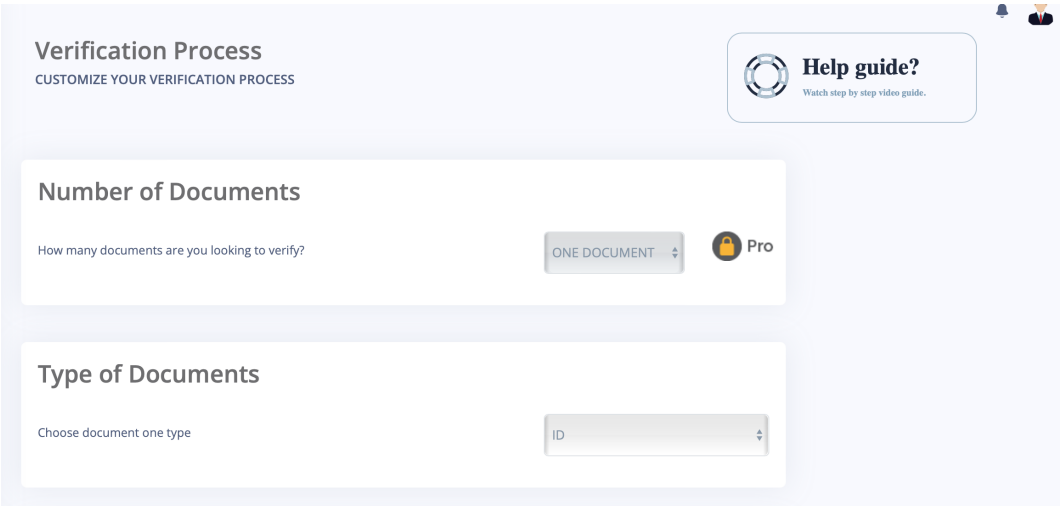


Figura 7.33: Configuración de documentos válidos para autenticación con KYC

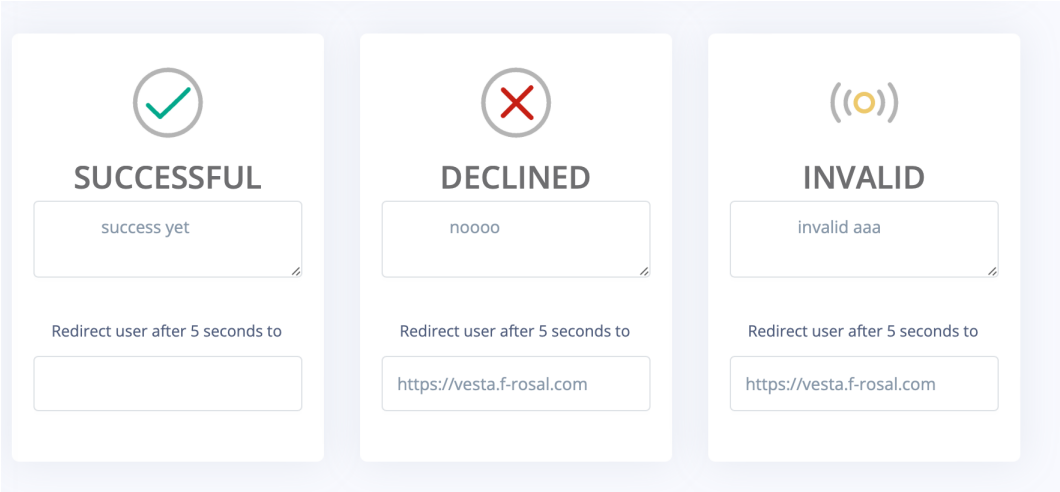


Figura 7.34: Configuración de enlaces a redirigir dependiendo de la autenticación con KYC

7.12. JSON *web token* y cifrado de información

Se implementó el JSON *web token* para la autorización en las diferentes peticiones que hagan los usuarios. Primero se tuvo que buscar sobre cómo implementar el JWT en la tecnología que se escogió para el *backend*, la cual es Ruby. Se encontró que existe la gema JWT para usarse en Ruby, la cual soporta varios algoritmos para poder crear y firmar los tokens. Para decidir cuál de los diferentes algoritmos usar para los tokens se realizó una investigación sobre cada uno de ellos para poder compararlos y ver la mejor opción [ver anexos B.1, B.2 y B.3]. Se decidió usar el algoritmo de RSA, ya que tenía las mayores ventajas para el sistema. Por un lado, nos ayuda que sea más fácil la distribución de las diferentes llaves, ya que únicamente se necesita repartir su llave privada a cada persona y en este caso también para cada ambiente de desarrollo. Por otro lado, este algoritmo nos ayuda a tener una escalabilidad mucho mayor, debido a que es mucho más fácil agregar nuevos usuarios porque solo hay que darles su llave privada. Por último, nos decidimos principalmente por este algoritmo,

ya que nos garantiza que provee autenticación y no repudio, lo cual es importante para los JWT. Además, que las llaves serán más seguras ya que son de una cantidad de *bits* mucho mayor a los otros algoritmos. La única parte negativa que se encontró fue que podía llegar a ser bastante demandante para los equipos, sin embargo, no era algo que preocupara y convenía más las ventajas que brinda.

Luego de haber definido el algoritmo se realizó la implementación del JWT, tanto para generar el token como para verificarlo. Primero se define el algoritmo a utilizar. En el archivo de credenciales se guarda las direcciones hacia la llave pública y privada, que se utilizará. En el método para generar el token se obtienen la llave privada y se crea el *payload* que se enviara con el token. Luego simplemente se genera el token con la llave privada, el *payload* y el *header* que contiene el algoritmo de cifrado. Para el método de descifrado simplemente se obtiene la llave pública del archivo de credenciales y se utiliza para descifrar el token y ver que si integridad no haya sido comprometida [ver anexo C.4].

Una vez ya se implementó y se comprobó que estuviera funciona nado se hizo un análisis de la entropía del token. Para esto se utilizó la herramienta de Burp [ver anexo B.4]. Esta herramienta se configuró para que escuchara las peticiones que se hicieran. Cuando se obtuvo la que genera el token, esta se seleccionó para que se use en el secuenciador, el cual se encarga de ejecutar múltiples veces esa petición y se configura para que analice la entropía de lo seleccionado en la respuesta, en este caso el token, se realizaron un total de 5000 peticiones para tener una gran cantidad de tokens que analizar Esto se hizo tanto para el algoritmo de HMAC como para el de RSA, y así poder comparar si la elección del algoritmo fue la correcta.

7.13. Análisis de riesgos y políticas de seguridad

Se realizó un análisis de riesgos para poder identificar posibles amenazas que puedan presentar un riesgo hacia nuestra plataforma. Para cada uno de los riesgos encontrados se determinó cuál es el impacto que podría llegar a tener en la plataforma en caso llegara a ocurrir. Además, se agregó una posible solución o contingencia por si se presenta el riesgo. Para determinar los distintos riesgos se preguntó a los desarrolladores de la plataforma sobre posibles amenazas en su módulo desarrollado para poder contemplar las distintas áreas que conforman el proyecto. Para también prevenir algunos de estos riesgos se establecieron diferentes políticas de seguridad las cuales nos ayudan a definir diferentes reglas, comportamientos o acciones que se deben seguir en todo momento para poder garantizar la seguridad de la plataforma y reducir la probabilidad de que alguna de las amenazas pueda ocurrir.

7.14. Validación legal

Una vez se desarrolló un mínimo producto viable, se validó con partes interesadas, especialmente del lado legal. Para esto, se entrevistó al Licenciado Federico G. Rosal Morales, Abogado con colegiado activo 7035, Oficial Mayor de la Corte Suprema de Justicia para saber su opinión y validar el marco legal del presente proyecto. Adicionalmente, se investigaron los procesos actuales que se realizan en instituciones como el Registro de la Propiedad y la Superintendencia de Administración Tributaria.

CAPÍTULO 8

Resultados

8.1. *Smart contract*

El *smart contract* desarrollado se encuentra desplegado tanto en la red de prueba de Görli, como en la *mainnet*. También, el *smart contract* es compatible con todas las redes de *blockchain* que utilizan la Ethereum *Virtual Machine* (EVM), de manera puede ser desplegado en muchas otras redes aparte de Ethereum y una de estas es la red de prueba de Avalanche, Fuji, en donde también se desplegó e interactuó con el contrato de forma exitosa (Ver anexo E).

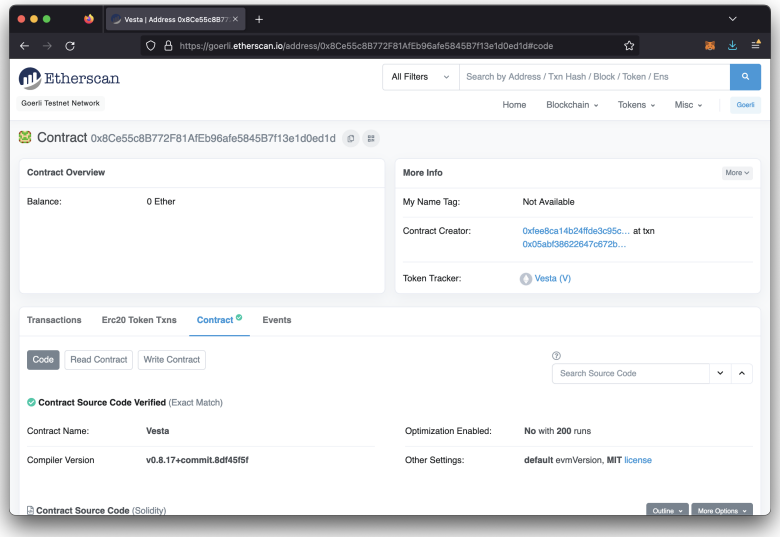


Figura 8.1: *Deploy* del *smart contract* en la *testnet* de Ethereum, Görli

Con el contrato desplegado en la red, todas las transacciones en las cuales esta involucrado el *smart contract* desarrollado, son evidenciadas por los exploradores de bloques como Etherscan, en donde el resultado principal es la transparencia de todas las operaciones efectuadas. Como se puede observar en la 8.2 para cada transacción se puede saber con que función del *smart contract* interactuó cierto usuario, así como el costo de la transacción.

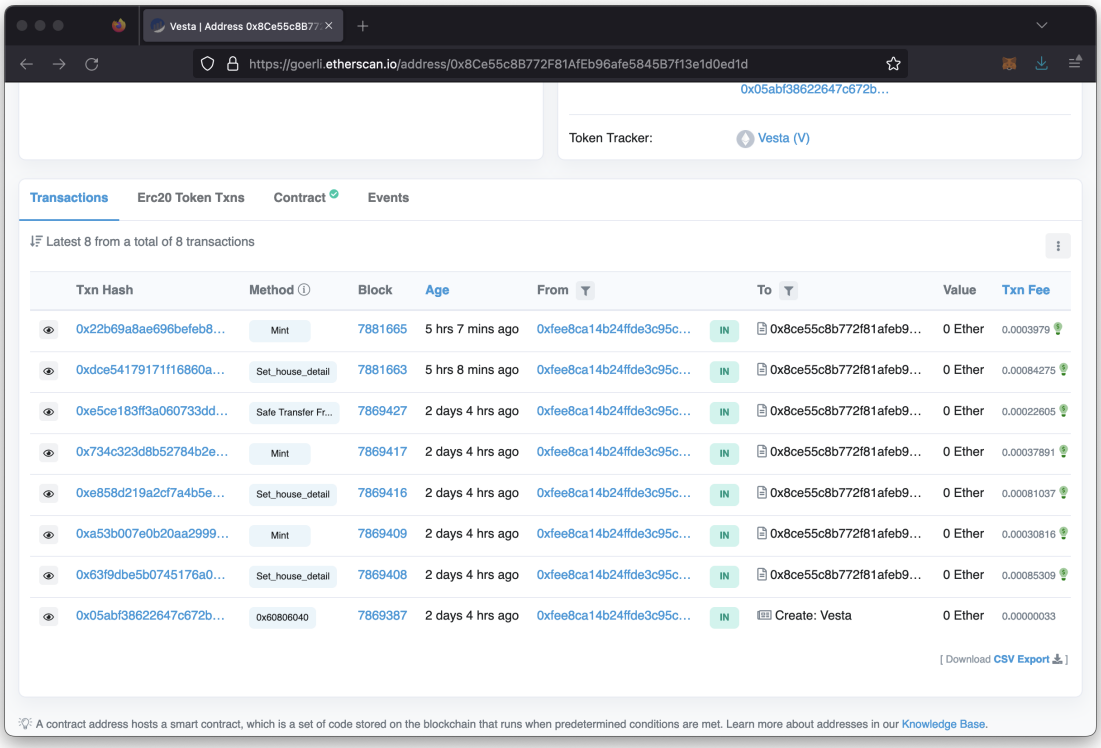


Figura 8.2: Transacciones realizadas por medio del *smart contract*

Además, para cada transacción se puede tener una vista aun más detallada como se puede observar en la Figura 8.3.

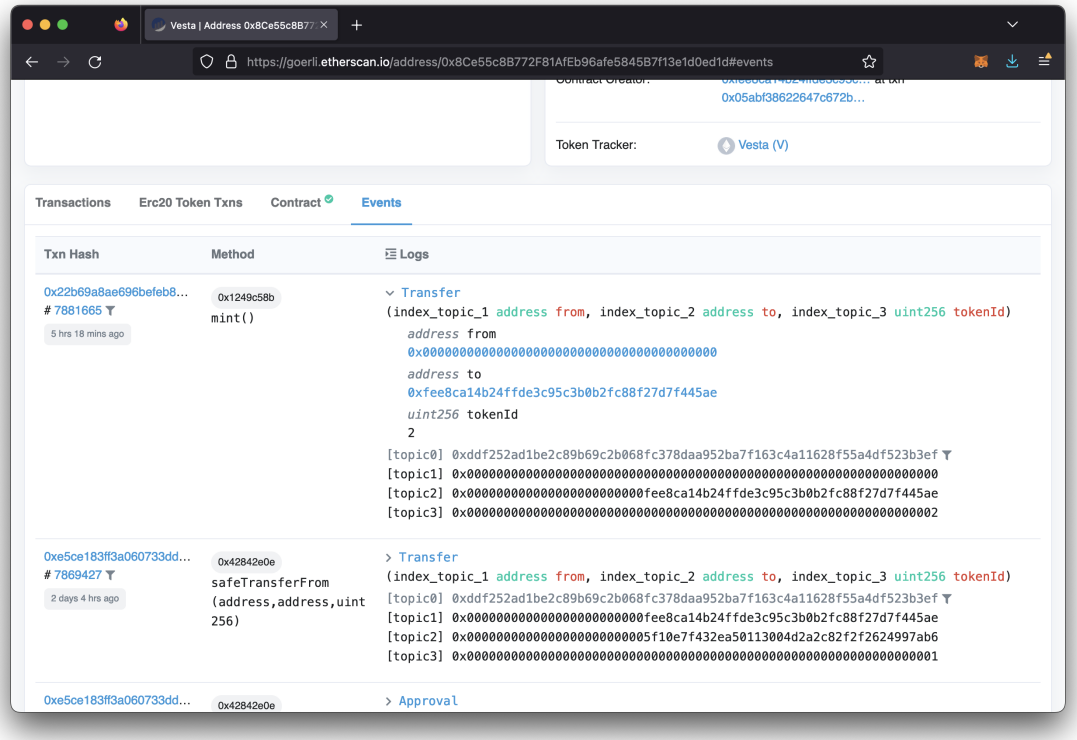


Figura 8.3: Transacciones detalladas realizadas por medio del *smart contract*

Debido a que el *smart contract* se desarrolló cumpliendo las reglas del ERC-721, todo *NFT* creado por el *smart contract* puede ser interpretado por la gran mayoría de plataformas de comercio de *NFTs*. En la Figura 8.4 se puede observar un *NFT* en la plataforma de OpenSea, y en la Figura 8.5 en la plataforma de Rarible.

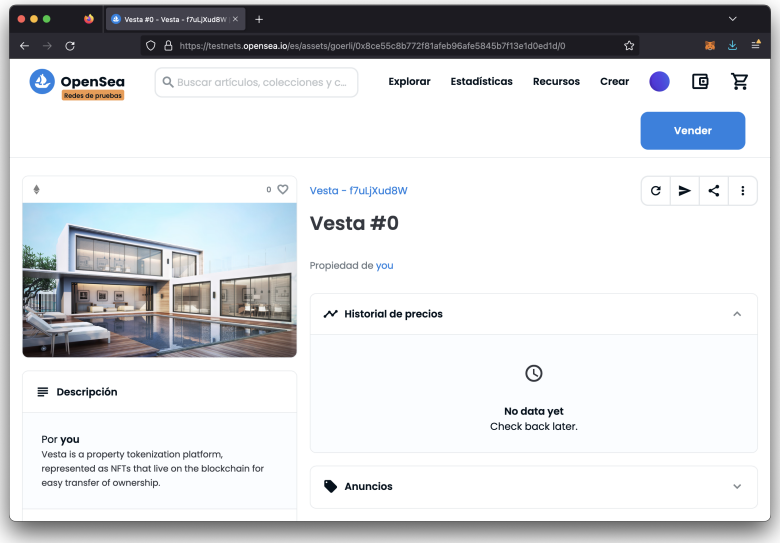


Figura 8.4: *NFT* listado en OpenSea

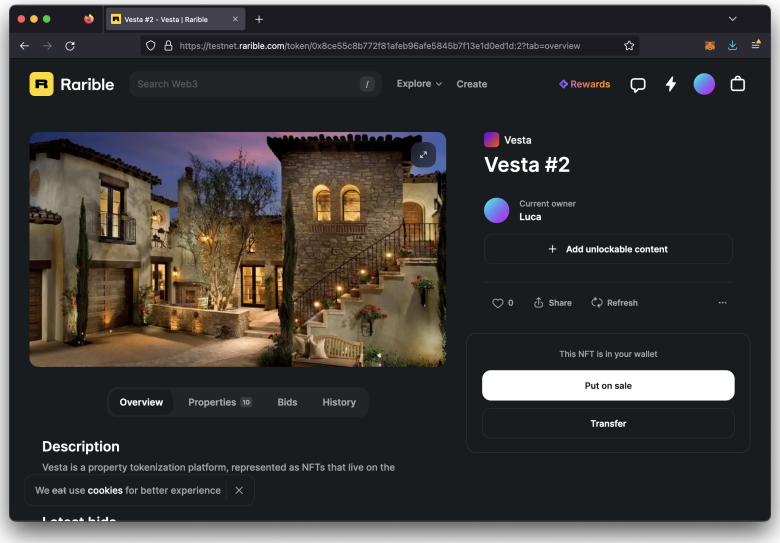


Figura 8.5: *NFT* listado en Rarible

La importancia de lograr que el *NFT* sea visualizado por otras plataformas de *NFTs* es demostrar los metadatos del mismo. Esto se logró de manera exitosa ya que los metadatos del *NFT* es almacenada en el mismo *smart contract* y es enviada según la estandarización de un *tokenURI* para poder ser leído por plataformas de terceros. En la Figura 8.6 se muestran los metadatos en la plataforma de OpenSea y en la Figura 8.7 en la plataforma de Rarible.

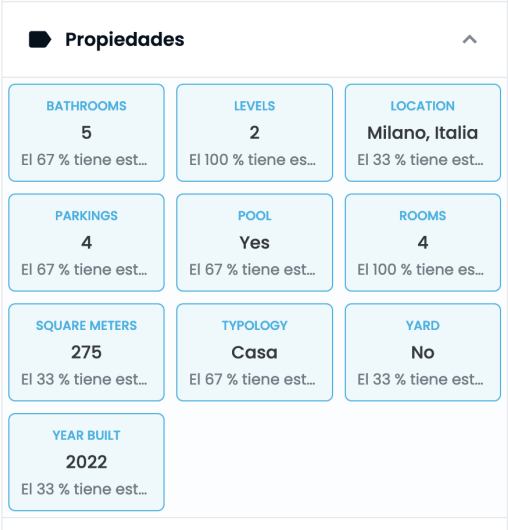


Figura 8.6: Metadatos del *NFT* en OpenSea

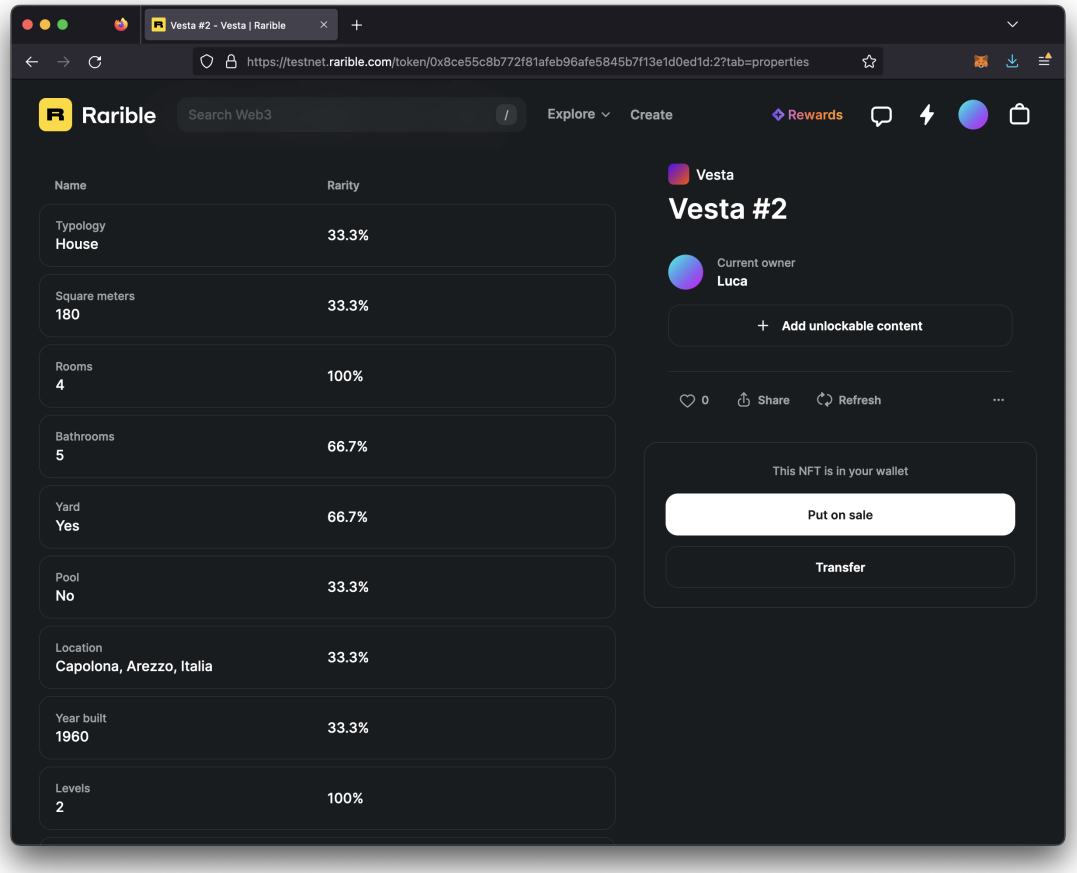


Figura 8.7: Metadatos del *NFT* en Rarible

8.2. *Web3*

El *smart contract* no solo tuvo la capacidad de producir *NFTs* basado en el ERC-721, sino que también pudo ser dinámico para la elaboración de una plataforma de usuario con el cual se comunica al contrato inteligente para ejecutar sus funciones. La plataforma se desarrolló con base en el diseño de *UI/UX* y se conectó al contrato por medio de *Web3.js*. En la plataforma se pueden observar los *NFTs* que han sido creados, así como la información de los metadatos de cada uno de los *NFTs* creados, crear nuevos *NFTs* y si se es el dueño de un *NFT*, se tiene la posibilidad de poder transferir la propiedad de un *NFT* como una representación del bien inmueble que ese *NFT* representa.

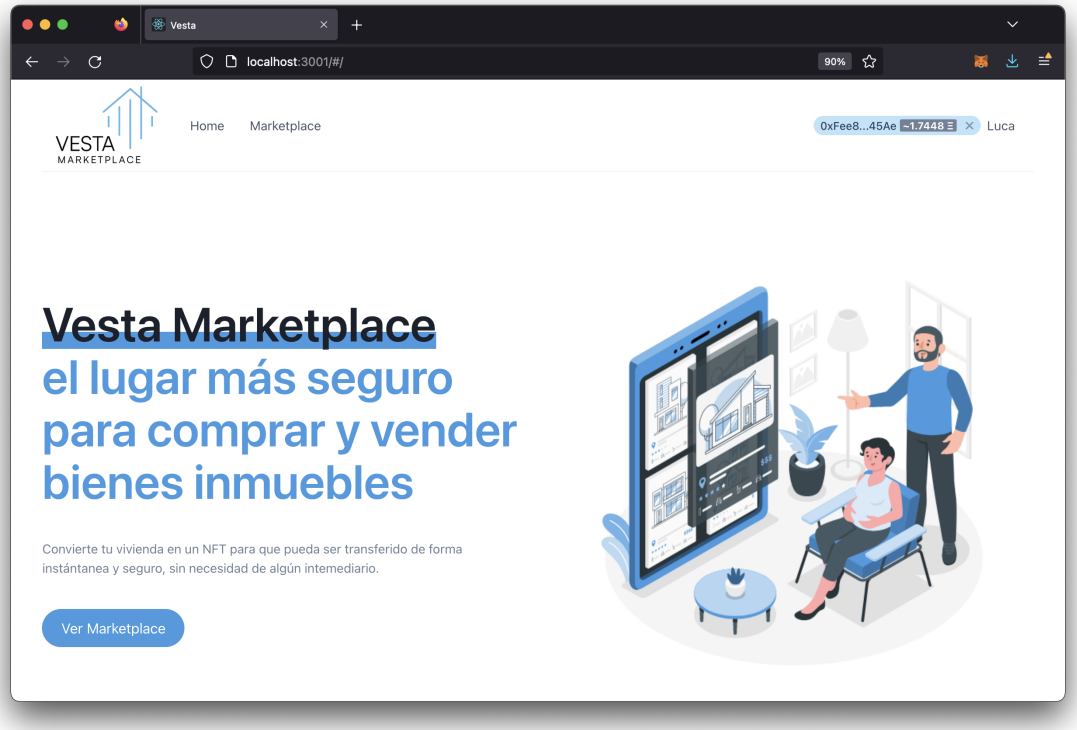


Figura 8.8: Página principal

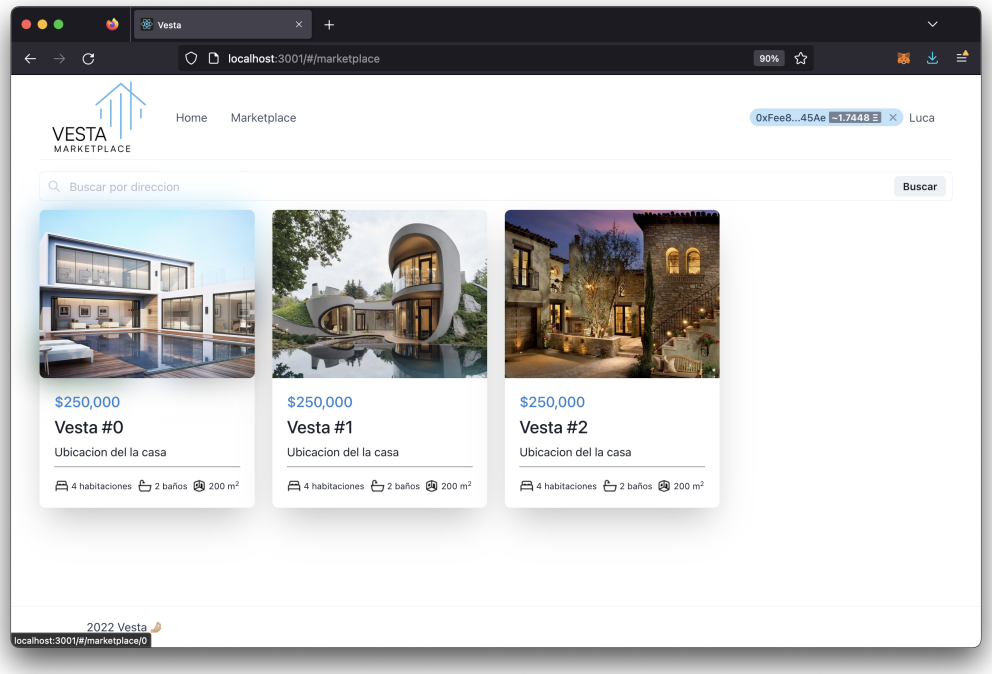


Figura 8.9: Página en donde se listan las propiedades creadas

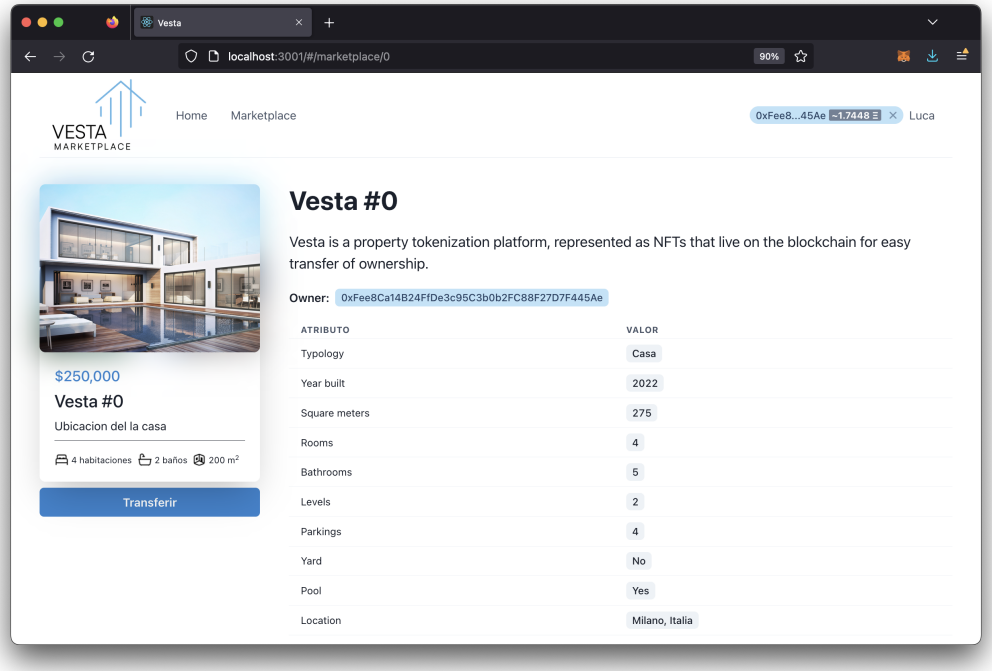


Figura 8.10: Página de visualización de una propiedad

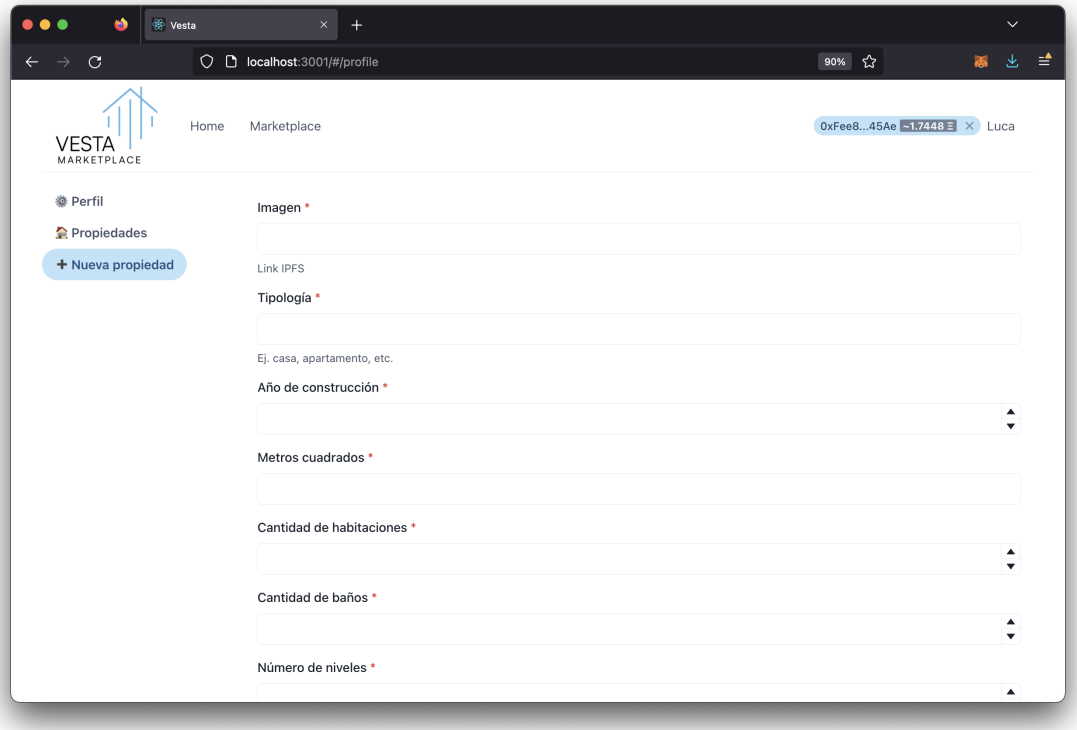


Figura 8.11: Página de creación de un nuevo *NFT*.

8.3. Base de datos

En cuanto al diseño de base de datos no es necesario realizar cambios drásticos; la implementación depende del diseñador de la base de datos. Sin embargo, según lo experimentado en la metodología, lo mínimo requerido sobre la base de datos es una tabla adicional donde se pueda almacenar las *wallets* que los usuarios pueden tener. Y adicionalmente, agregar un campo de identificación en las entidades que también se almacenan en la red para poder relacionarlas. El resultado final de la implementación de la base de datos se puede visualizar a continuación.

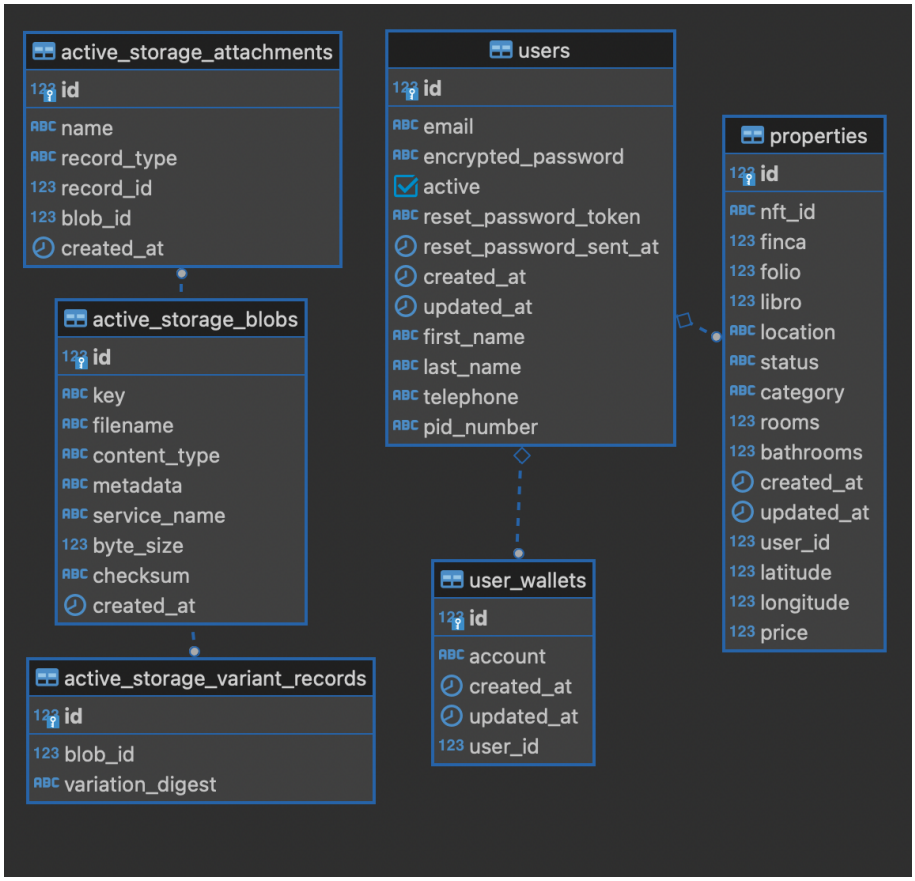


Figura 8.12: Resultado del diagrama de entidad-relación.

8.4. Servidor

Una aplicación de *blockchain* puede funcionar fácilmente sin un *backend* tradicional, porque la red de *blockchain* es el equivalente a la base de datos y los *smart contracts* que almacena actúan como *backend* proporcionando un API para su uso; de este modo el *frontend* puede consultar los *smart contracts* usándolos como API para poder acceder a la información.

Sin embargo en este proyecto se habla sobre una integración entre *blockchain* y *backend*. Esta relación de integración es valiosa cuando se opera en redes *blockchain* con muy alto costo transaccional y de almacenamiento. Adicionalmente, una red *blockchain* usualmente se caracteriza por su anonimato, ya que una sola persona puede conectarse a la misma red con diferentes cuentas o *wallets*. Entonces para el caso de uso específico que se trata en este proyecto que es el de un sistema de títulos de propiedad, el anonimato es una característica que no aporta valor. De este modo, una integración con un *backend* tradicional tiene cierto atractivo, ya que se puede proveer identidad a los usuarios y controlar con facilidad ciertos aspectos, como se hace en un sistema centralizado. Además se reduce en gran cantidad la información que se almacena en la red, lo cual implica reducción de costos transaccionales y de almacenamiento para cada nodo participante. Lo que resulta muy atractivo en una red.

A nivel de la infraestructura de nube no existe mayor variación que se deba implementar para hacer posible esta integración de tecnologías. Sin embargo hay que considerar que la infraestructura de nube debe poder soportar el nivel de concurrencia que existe en la red. En este caso la concurrencia es relativamente baja, ya que la compra-venta de inmuebles no es una acción que ocurra constantemente o en masa. Sin embargo, en otras aplicaciones donde el propósito es una alta concurrencia como lo sería una red social, se debería contemplar en el momento de diseño, selección de tecnologías e implementación de la infraestructura de nube. El soporte de concurrencia de la infraestructura debe ser directamente proporcional al de la concurrencia en la red.

A continuación se puede visualizar las pantallas principales de https://vesta.f-rosal.com que comprenden el resultado final de la implementación de infraestructura de nube, API y *frontend* con *Web3*.

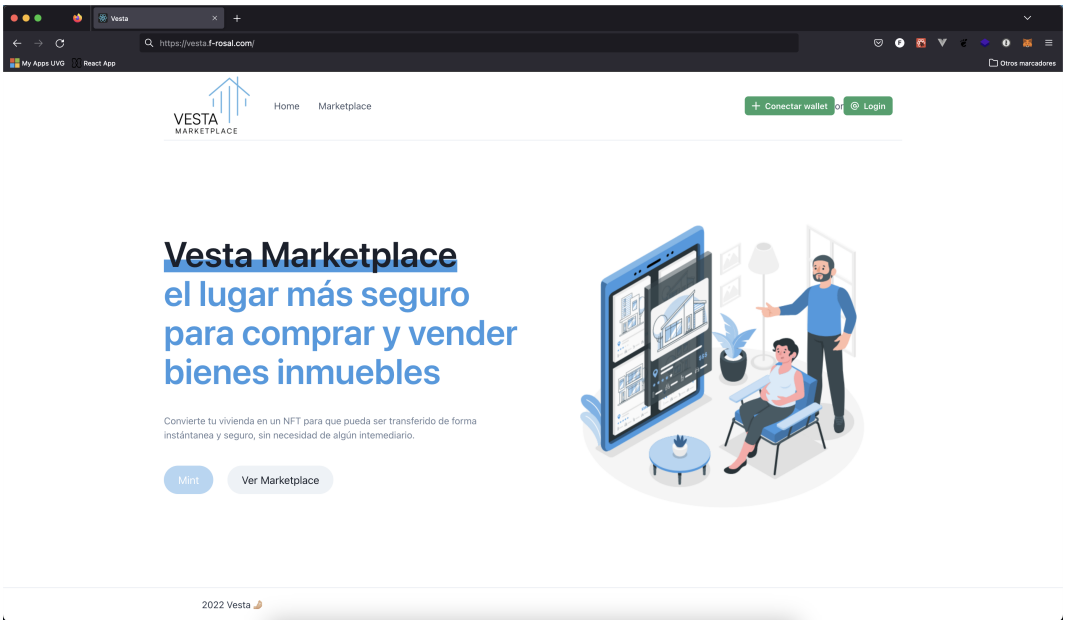


Figura 8.13: Resultado de la aplicación web - *Home*.

Figura 8.14: Resultado de la aplicación web - *Logged in*.

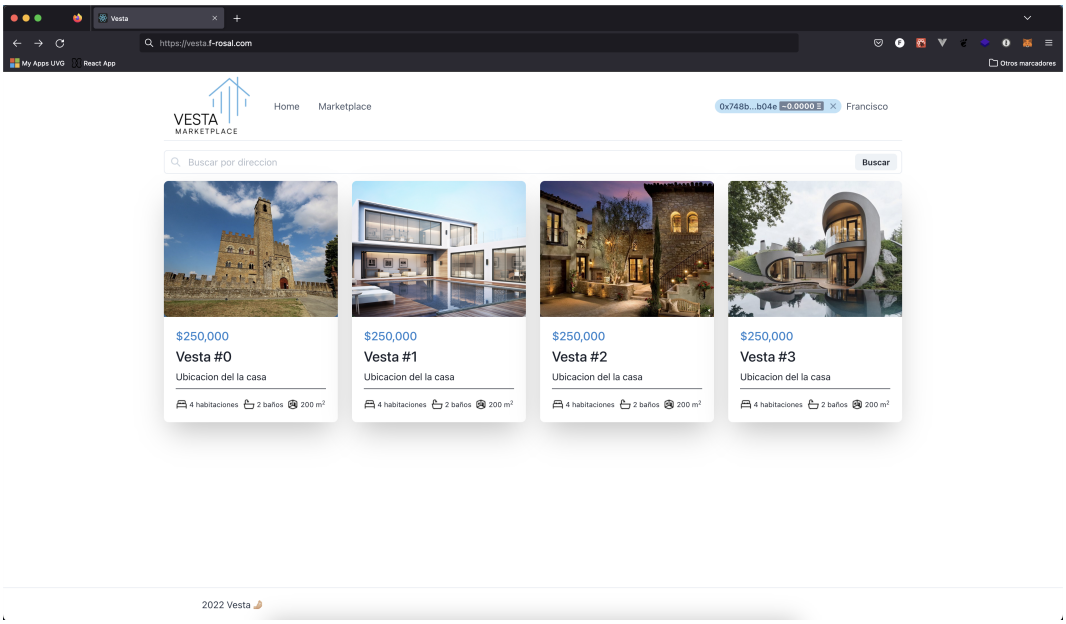


Figura 8.15: Resultado de la aplicación web - *Marketplace*.

8.5. CI

La implementación de *testing* e Integración Continua le agrega valor indirecto a la seguridad del *software*, ya que se implementaron flujos que evalúan diferentes aspectos de la aplicación. Los resultados de la implementación de CI se pueden visualizar en las imágenes proporcionadas a continuación. Entre las imágenes se puede visualizar diferentes *jobs* del *workflow* implementado y momentos en los que fue exitoso y fallido. Así mismo se puede ver como prohíbe el *merge* de *PRs* que los *jobs* detectan con posibles errores. Esto facilitó el desarrollo ya que no se encontraron errores no deseados durante su desarrollo. Además se adjuntan algunas imágenes de su configuración.

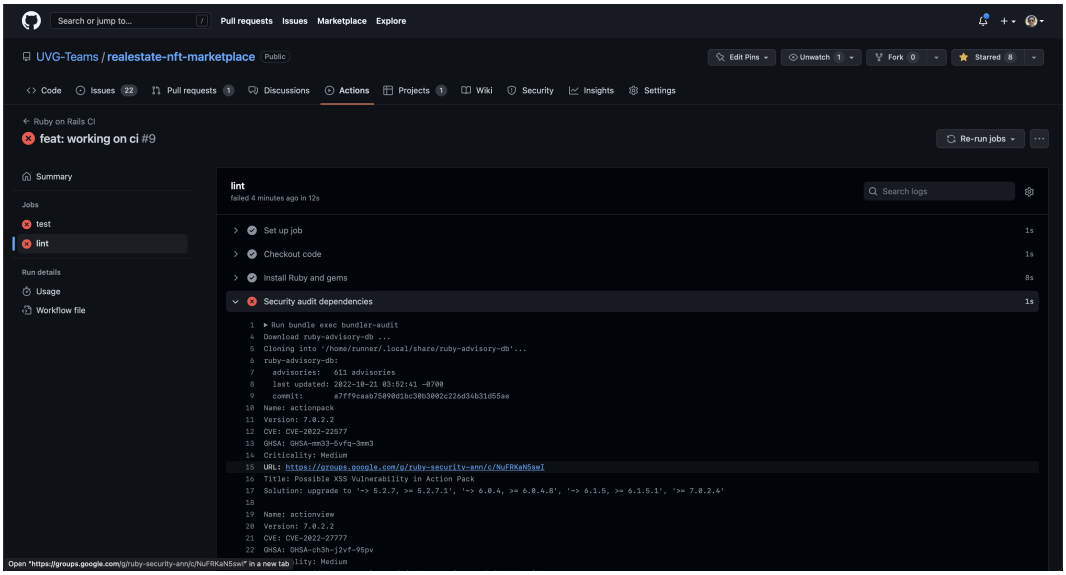


Figura 8.16: CI - 1.

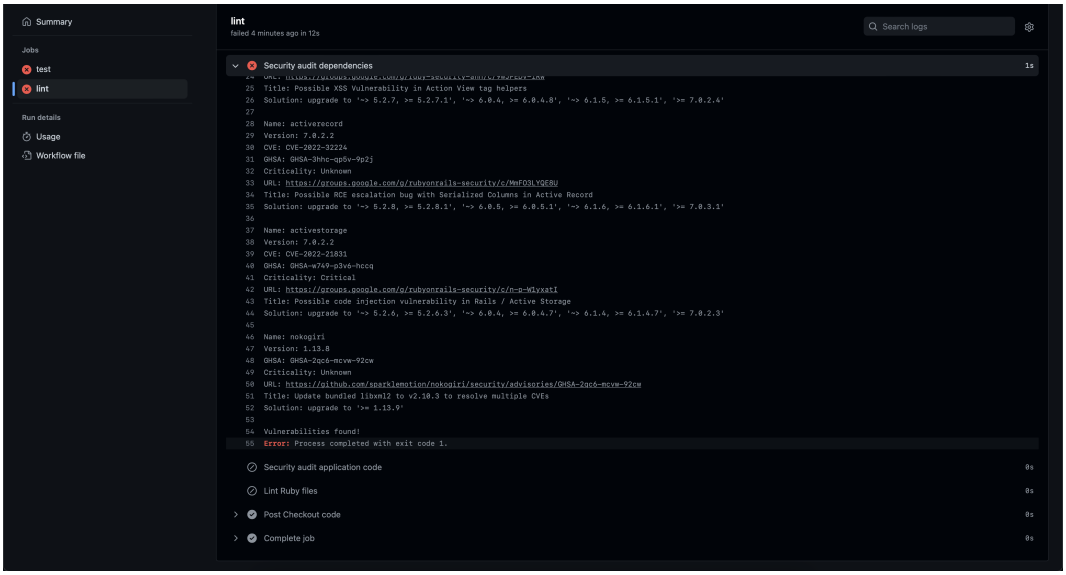


Figura 8.17: CI - 2.

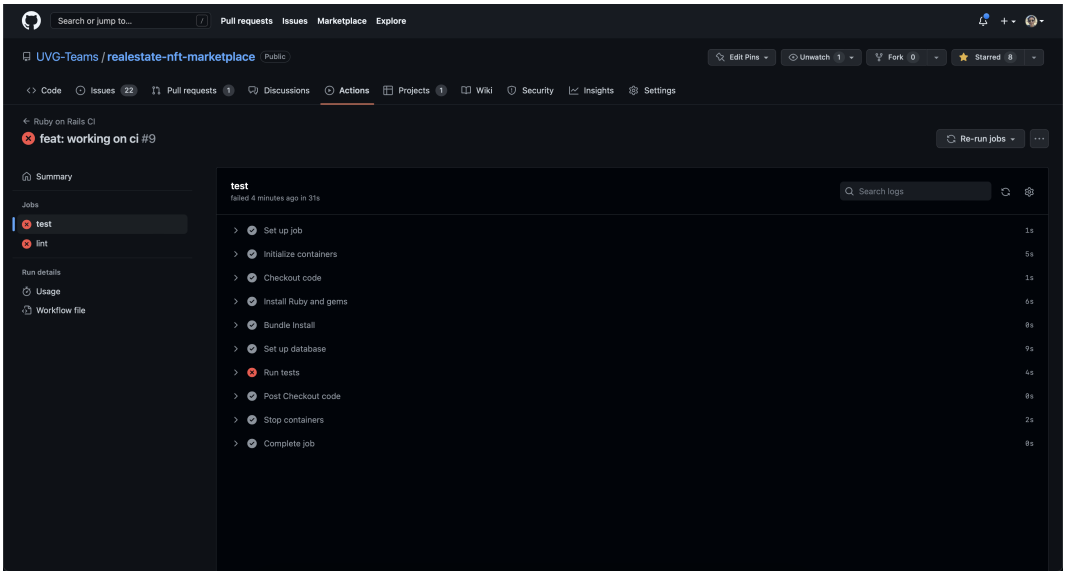


Figura 8.18: CI - 3.

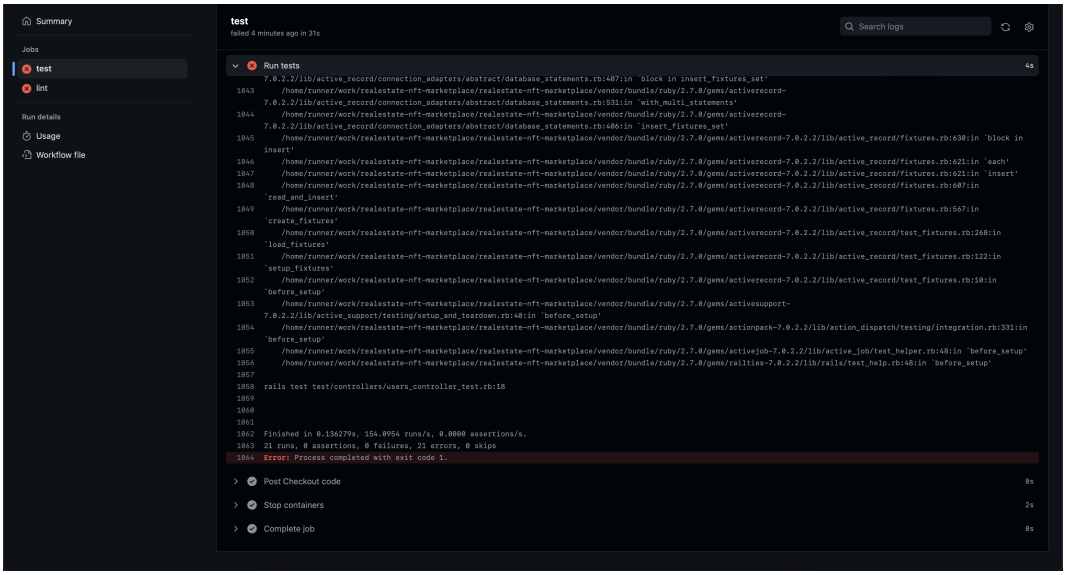


Figura 8.19: CI - 4.

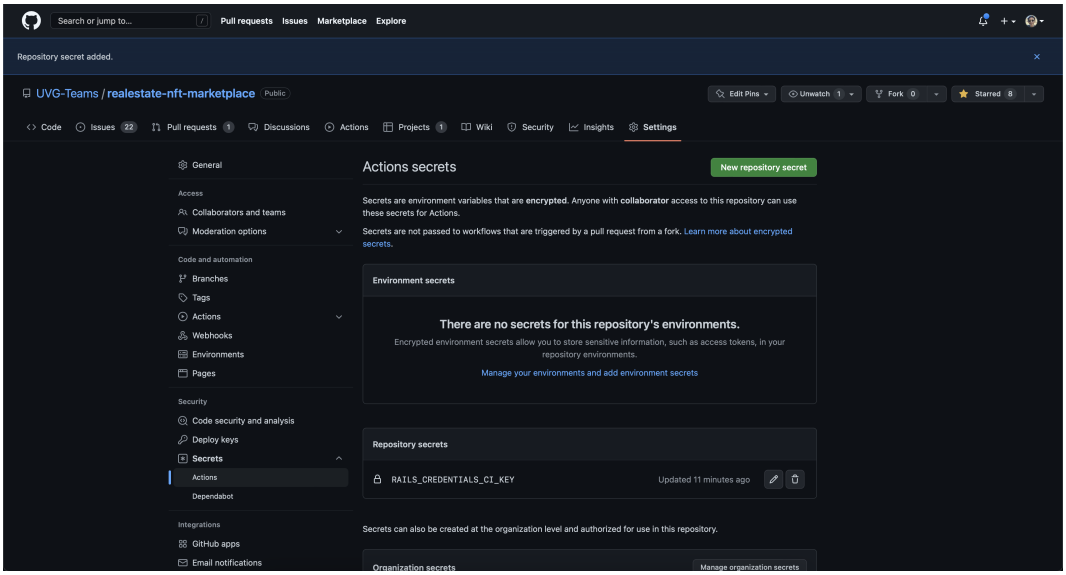


Figura 8.20: CI - 5.

Figura 8.21: CI - 6.

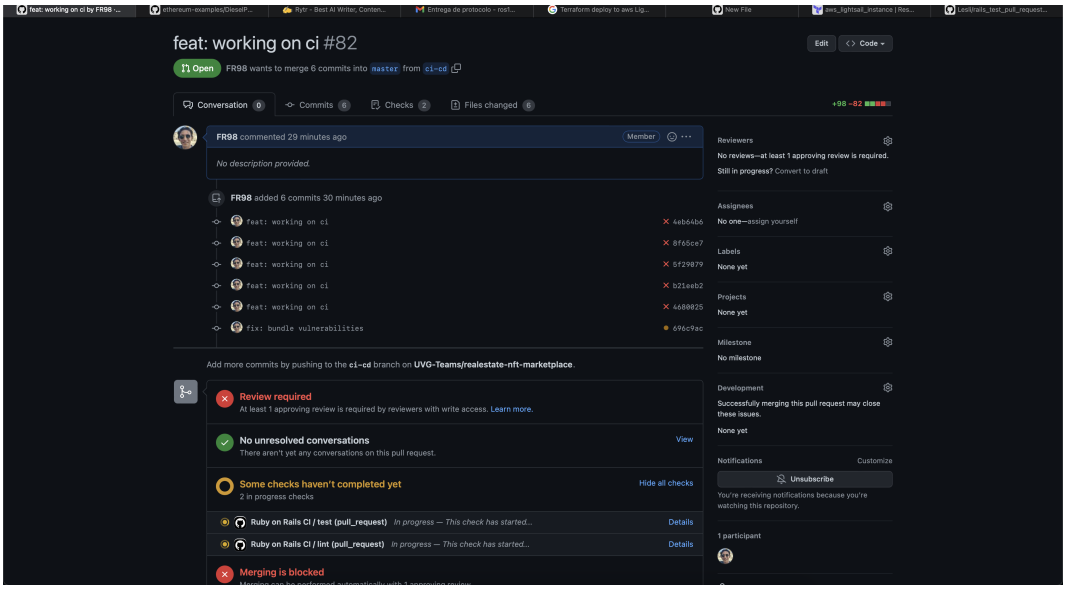


Figura 8.22: CI - 7.

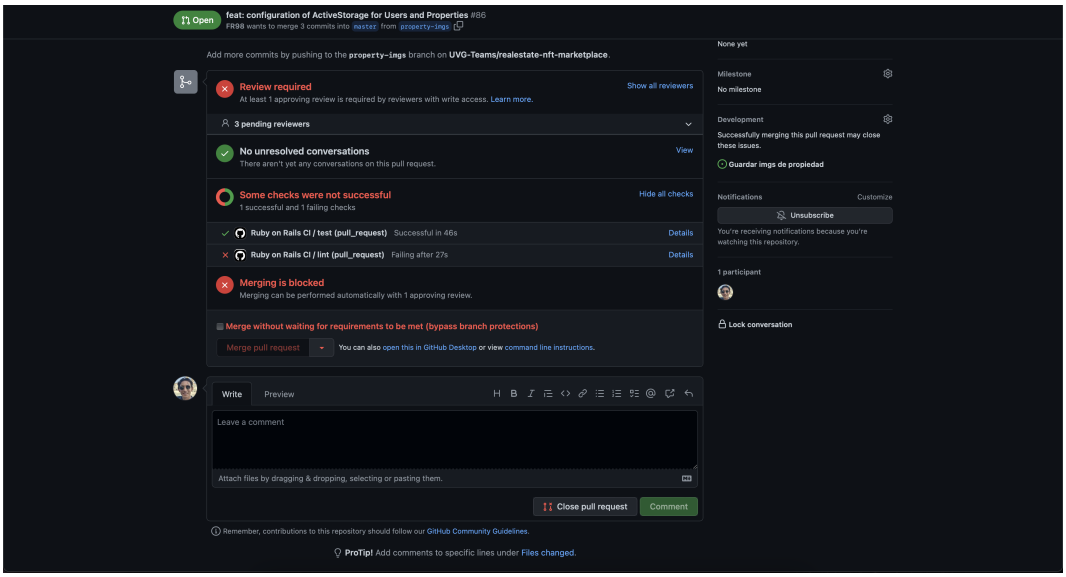


Figura 8.23: CI - 8.

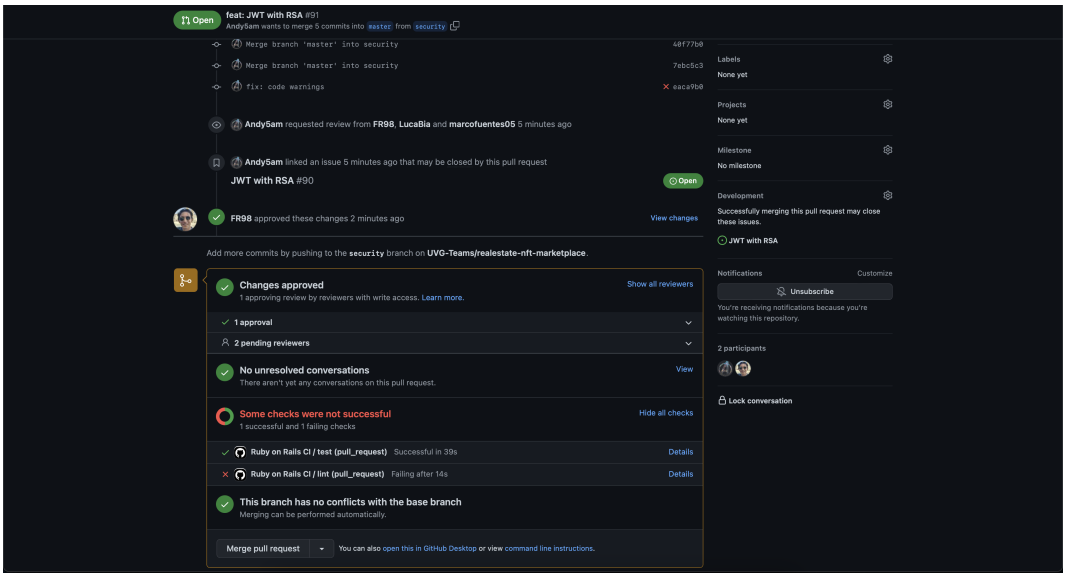


Figura 8.24: CI - 9.

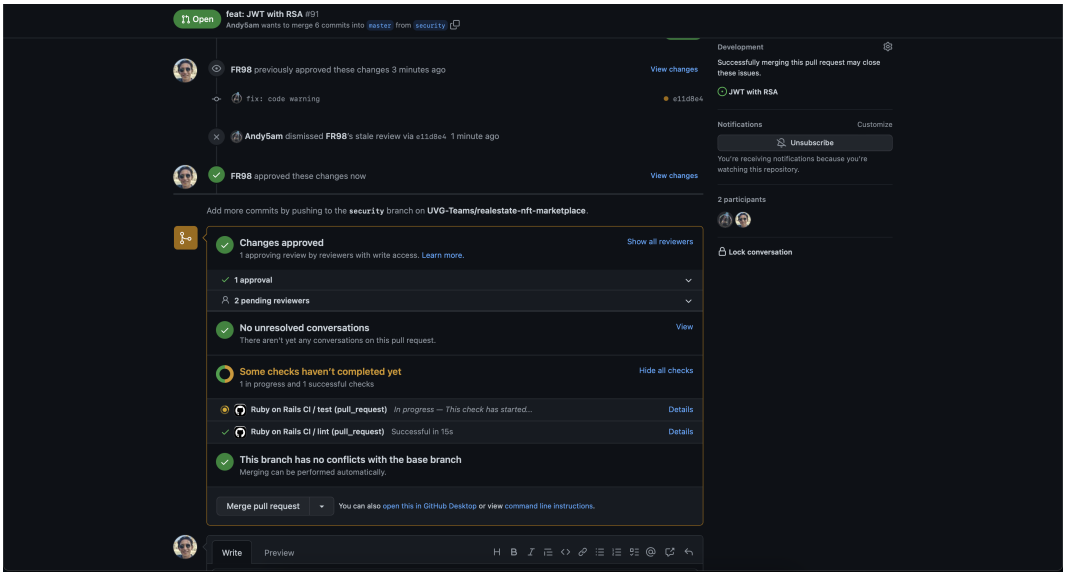


Figura 8.25: CI - 10.

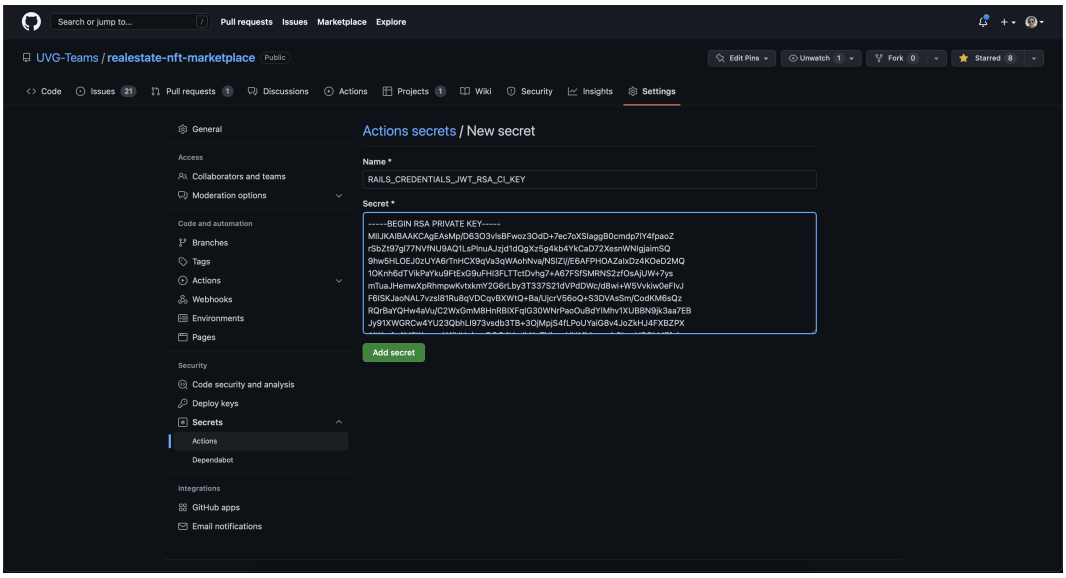


Figura 8.26: CI - 11.

8.6. Reconocimiento de imágenes

Ante el problema de reconocer y clasificar imágenes como parte o no de uno de los ambientes definidos de una casa se presentaron dos posibles soluciones.

Una de ellas era usar un acercamiento de una red neuronal convolucional multiclase, que tomara una imagen y devolviera un vector normalizado de probabilidades, en el cual cada entrada correspondería a un posible ambiente, y la entrada con mayor valor indicaría un reconocimiento.

Por otro lado, se propuso un acercamiento basado en varias redes neuronales convolucionales especializadas en un solo ambiente, a las cuales sería ingresada la misma imagen al mismo tiempo y sería considerado como reconocimiento si un resultado superara un umbral definido, y las demás no lo superaran.

8.6.1. Modelo multiclase

Como se puede apreciar en la Figura 8.27, luego de evaluar el modelo por 150 rondas se obtuvo una precisión en el conjunto de entrenamiento superior al 80 % y un valor de pérdida cercano a 0.05. Tomando en cuenta que la pérdida inicial era de 4.0, parecería que el modelo hizo un muy buen trabajo con el conjunto de datos que le fue proporcionado. Sin embargo, evaluar el comportamiento de la precisión en el conjunto de pruebas lleva a la conclusión opuesta. La precisión del modelo en datos de prueba se estabilizó cerca de 0.45. Este valor no fue el ideal pues, si bien es cierto que es un rendimiento mejor que el que tendría un sistema completamente aleatorio (que en este caso tendría 20 % de efectividad, al solo tener 5 opciones a escoger), siguió reportando más fallos que aciertos.

Otro problema que presenta este modelo es que no es consistente ante el rechazo. Al tener una capa de salida con una función de activación de tipo *softmax*. Si la red recibe una imagen que no debe pertenecer a ninguna clase, es incapaz de devolver un vector "0"de manera directa, pues gracias a que la capa de salida debe cumplir con ser una distribución de probabilidad, el *peor* resultado que podría devolver sería un vector en el que todos sus valores sean exactamente 0.2. Sin embargo, este resultado fue inconcluso. Un cliente de este modelo, si recibe este resultado, no tiene manera de saber si el resultado es un rechazo de todos los ambientes, o que la imagen corresponde a todos los ambientes al mismo tiempo.

Figura 8.27: Efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación multiclase.

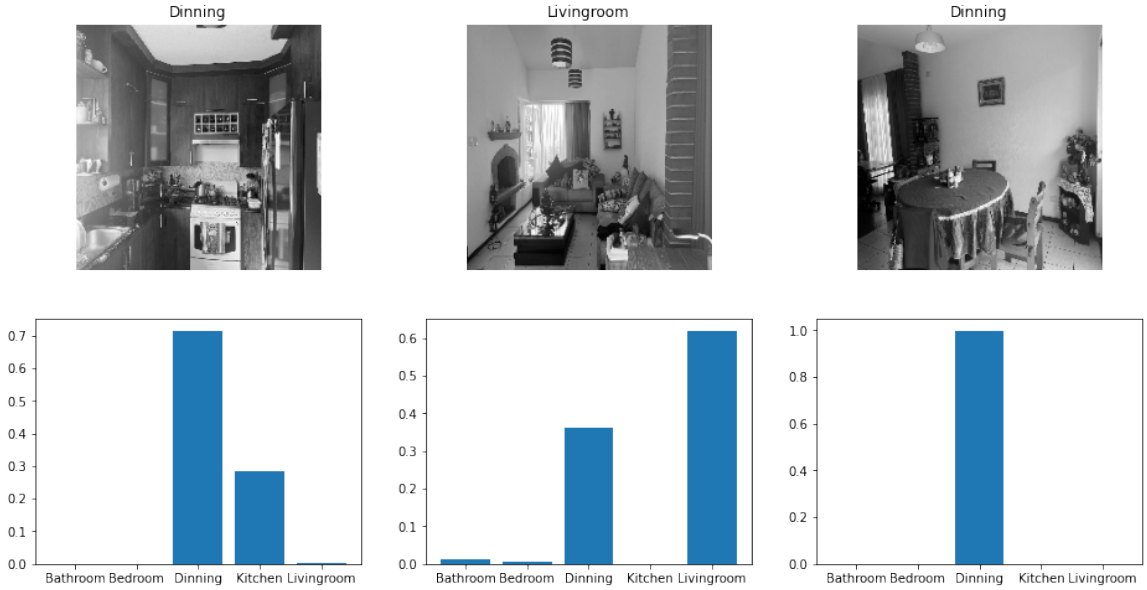


Figura 8.28: *Output* de la red multi clase al ser presentada con varias imágenes de casas reales.

8.6.2. Modelos binarios

Si bien es cierto que la propuesta de tener varios modelos binarios suena más laboriosa y menos eficiente que su contraparte multiclase, los resultados de ésta no fueron muy alentadores y parecieran indicar que la mejor solución a este problema vendrá con clasificadores binarios trabajando de manera conjunta.

Modelo de clasificación de sanitario

Este modelo presentó, como se puede notar en la Figura 8.29, este modelo sí es capaz de mostrar una consistencia mayor entre resultados en el entorno de entrenamiento y de pruebas, a pesar de

contar con una cantidad de datos de entrenamiento considerablemente menor. Este resultado podría atribuirse a la presencia de la capa de regularización en la arquitectura de la red, justo en medio de las capas densas que se encargan de interpretar los hallazgos de las capas convolucionales. Sin embargo, es importante mencionar también que, al ser un problema más simple (*simple* en el sentido en que solo hay dos posibles salidas, en lugar de 5), el modelo parece ser capaz de especializarse y aprovechar al máximo sus recursos disponibles para encontrar una mejor solución al problema planteado, en este caso la clasificación de la imagen en una de dos categorías: sanitario y no-sanitario.

Además de evitar el problema de sobreajuste, el modelo termina con una precisión de validación cercana a 80 %. Con estos resultados fue posible armar una matriz de confusión y ajustar el umbral que este modelo tendría para funcionar dentro del sistema de validación de datos. La matriz de confusión para este modelo se puede apreciar en la Figura 8.30.

A pesar de presentar resultados positivos en ambientes de prueba y de entrenamiento, este modelo presenta limitaciones en algunos ambientes de la vida real. Especialmente en aquellos donde la imagen puede verse distorsionada por haber sido tomada con un lente angular, o en los que los fondos o muros contienen muchos detalles. Estos puede causar que el modelo interprete patrones irrelevantes como activaciones de sus filtros convolusionales y, por tanto, el resultado sea distinto al esperado. Un ejemplo de esto se puede observar en la Figura 8.31, en donde únicamente una imagen de un sanitario es correctamente clasificada, y las otras dos son descartadas.

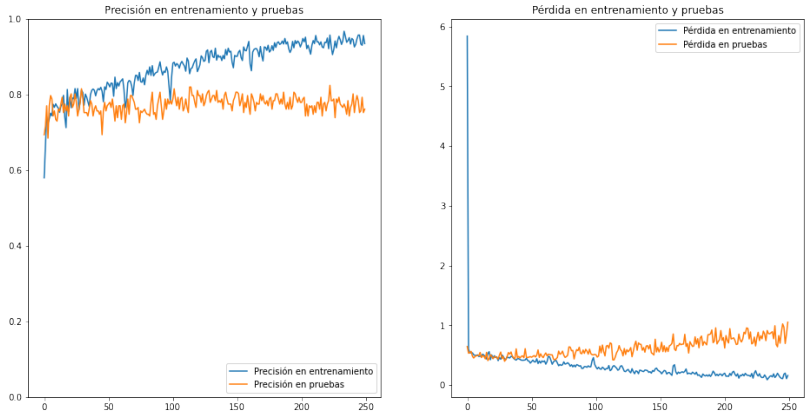


Figura 8.29: Efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de sanitario.

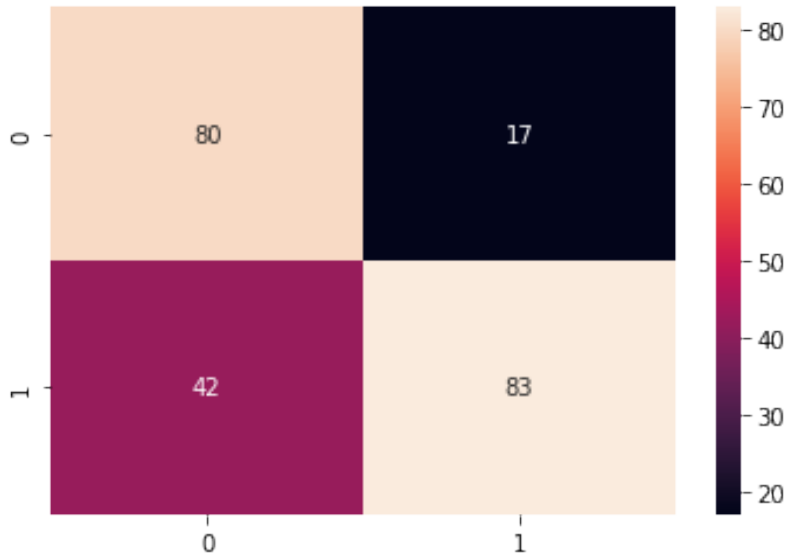


Figura 8.30: Matriz de confusión del modelo de sanitario al ser evaluado con imágenes de validación posterior al entrenamiento con un valor de umbral de 0.9.

Figura 8.31: Resultado del modelo de clasificación de sanitarios al recibir imágenes de ambientes reales.

Modelo de clasificación de dormitorio

Este modelo tuvo una efectividad similar a la de su modelo homólogo de sanitarios, como se puede observar en la Figura 8.33, como era de esperarse al compartir su arquitectura. Además, tanto el ambiente de sanitario como este de dormitorio coinciden con que son los ambientes menos saturados dentro del conjunto de datos en cuanto a elementos visuales concierne. Un ejemplo de esto es la Figura 8.32, en donde es posible notar que los elementos más importantes que se distinguen de la habitación son la cama, las almohadas y las ventanas, mientras que en el sanitario se tiene que los elementos de mayor relevancia son el inodoro, el espejo y el lavamanos. Claramente son ambientes diferentes, y los modelos deben ser capaces de diferenciarlos, pero tiene sentido que compartan su arquitectura si se conoce que son ambientes poco saturados en comparación con los demás ambientes, algunos de los cuales se caracterizan por tener una gran cantidad de elementos presentes (como

utensilios en una cocina, o platos y sillas en un comedor).

Luego de analizar los resultados se descubrió que el mejor valor umbral para este modelo fue 0.5. La matriz de confusión para este modelo con este umbral puede ser vista en la Figura 8.33.

De la misma manera que el modelo de sanitarios, este modelo presenta limitaciones al ser presentado con imágenes de ambientes reales, en particular cuando las imágenes fueron capturadas con lentes angulares o bien cuando las imágenes contienen una gran cantidad de elementos visibles. Estos generan ruido a nivel de píxeles y pueden causar que los filtros de la red no se activen de la manera en que se esperaría. Un ejemplo claro de esto se evidencia en la Figura 8.35, en donde la imagen No. 2 de la primera fila debió haber sido clasificada como dormitorio, pero por la alta concentración de elementos distractores puede estar causando problemas a los filtros, junto con la deformación causada por el lente angular de la cámara usada para capturar la fotografía. La imagen No. 3 de la primera fila la Figura 8.35 fue tomada en el mismo dormitorio, pero con un lente no angular y eliminando objetos distractores de la composición.

Figura 8.32: Comparación de saturación visual entre ambientes de dormitorio y sanitario.

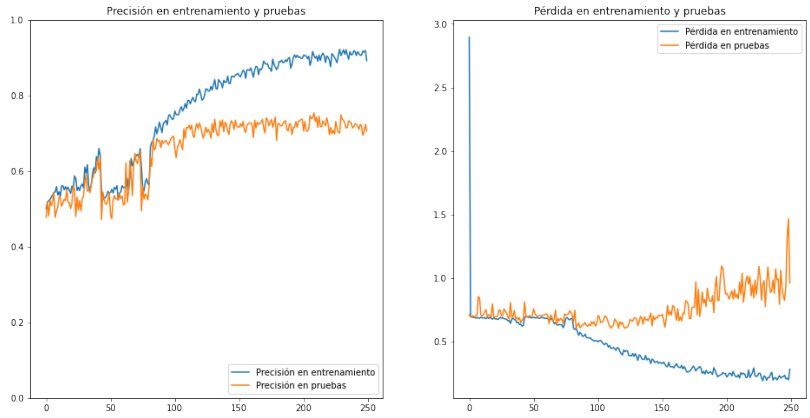


Figura 8.33: Evolución de efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de dormitorio.

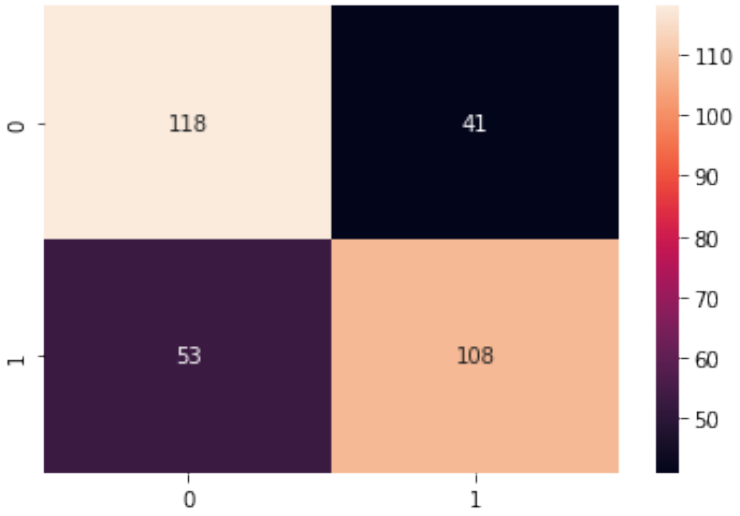


Figura 8.34: Matriz de confusión del modelo de dormitorio al ser evaluado con imágenes de validación posterior al entrenamiento con un valor de umbral de 0.5.

Figura 8.35: Resultado del modelo de clasificación de dormitorios al recibir imágenes de ambientes reales.

Modelo de clasificación de cocina

Este modelo también alcanzó buenos resultados, como se puede notar en la Figura 8.36, similares a aquellos de los modelos de dormitorio y sanitario. En este caso, la arquitectura tuvo que ser aún más profunda para que el resultado fuera satisfactorio. Tuvo que ser así porque típicamente en las cocinas en las casas se encuentran una gran cantidad de herramientas y utensilios, de los cuales muchos pueden servirle al modelo para identificar que se encuentra (o no) en un ambiente de cocina.

Los resultados mostrados en la Figura 8.37 muestran que, con un valor umbral de 0.8, este modelo lograba predecir correctamente en la mayoría de los casos. Sin embargo, en la Figura 8.38 también es posible notar que, a pesar que se colocaron dos fotografías de la misma cocina (Fotografías No. 2 y 3 de la segunda fila), solo una fue reconocida. Esto pudo ser causado por los parámetros impuestos

a la capa de preprocesamiento de datos que antecede a la red neuronal y que se encarga de hacer modificaciones espaciales a los píxeles de las imágenes. Además, el modelo reconoció una fotografía de una calle (fotografía No. 1 de la segunda fila) como cocina. Esto se debe a que el conjunto de datos usado para el entrenamiento no contaba con imágenes de todo tipo de ambientes, este problema no estaría presente si los modelos hubieran sido entrenados con un *dataset* considerablemente mayor.

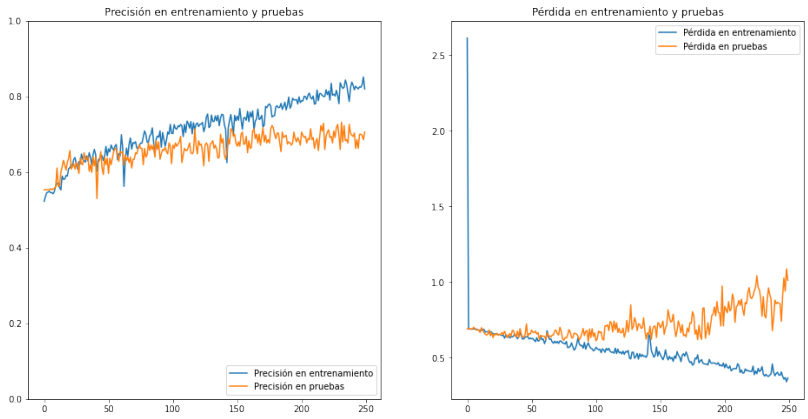


Figura 8.36: Evolución de efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de cocina.

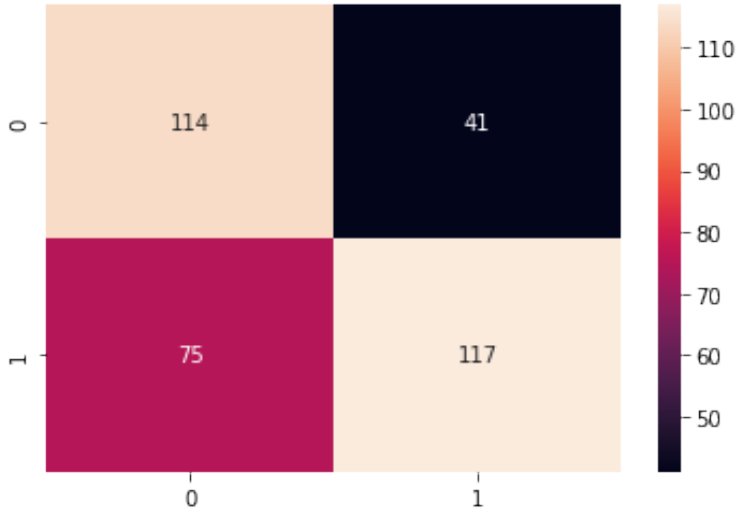


Figura 8.37: Matriz de confusión del modelo de cocina al ser evaluado con imágenes de validación posterior al entrenamiento con un valor de umbral de 0.8.

Figura 8.38: Resultado del modelo de clasificación de cocina al recibir imágenes de ambientes reales.

Modelo de clasificación de comedor

Como se puede ver en la Figura 8.39, este modelo presentó una precisión de aproximadamente 75 % al evaluar imágenes del conjunto de validación. En este modelo se aprovecharon los beneficios de la arquitectura del modelo de sanitarios y solo fue necesario hacer ajustes menores a la capa densa que precede a la de salida. La Figura 8.40 muestra la matriz de confusión del modelo después de haber completado la totalidad de las rondas de entrenamiento. Con esto se decidió que el umbral apropiado para este modelo era 0.85.

La modificación al modelo original fue necesaria porque luego de observación y rondas extra de entrenamiento se observó que el modelo tendía a sobre ajustar los resultados al conjunto de entrenamiento, y la mejor solución fue reducir la cantidad de neuronas densas que se encontraban en la última capa antes de la capa de salida. Reducir la dimensionalidad de los filtros convolusionales no era una buena opción porque la misma experiencia y resultados de modelos anteriores indicaban que el reconocimiento de propiedades (o *features*) estaba funcionando correctamente, por lo que el problema tenía que estar en el procesamiento de los filtros.

Este modelo también presentó limitaciones al presentarle imágenes de ambientes reales y otras de la vida real que no pertenecen a ningún ambiente de una propiedad inmobiliaria. En la Figura 8.41 se puede notar como, si bien el modelo reconoce perfectamente la imagen No. 1 de la segunda fila, también reconoce como comedor la fotografía de la cocina (segunda imagen de la segunda fila), y la fotografía de la calle que también hizo confundirse al modelo de cocinas.

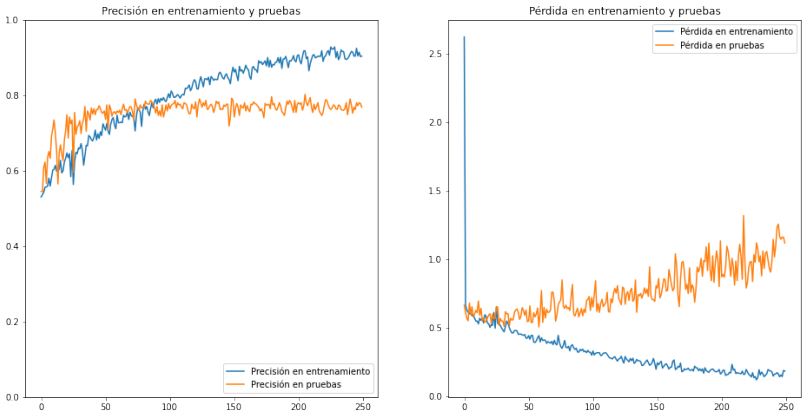


Figura 8.39: Evolución de efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de comedor.

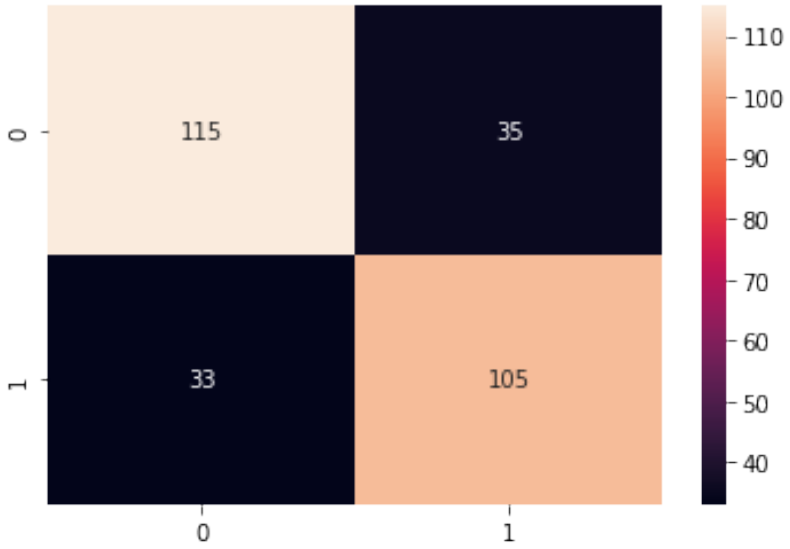


Figura 8.40: Matriz de confusión del modelo de comedor al ser evaluado con imágenes de validación posterior al entrenamiento con un valor umbral de 0.85

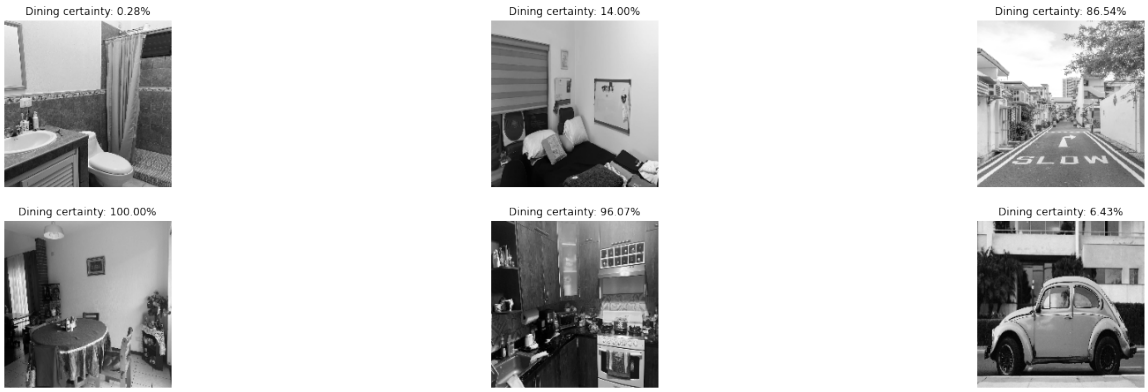


Figura 8.41: Resultado del modelo de clasificación de comedor al recibir imágenes de ambientes reales.

Modelo de clasificación de sala

En este caso no fue posible utilizar una arquitectura similar a la usada con el resto de modelos. El problema de clasificación de salas es considerablemente más complejo que los anteriores porque la estructura de una sala comparte muchas características con otros ambientes; una sala usualmente tiene una mesa central, sillas o sillones rodeándola (asemejando un comedor), o bien puede tener elementos decorativos colgando en las paredes tales como cuadros o repisas (asemejando una cocina, que también puede asemejar un comedor porque tiene mesas, sillas, etc.). Tantos elementos presentes y dispersos en todas las imágenes del conjunto de entrenamiento hicieron que un modelo de complejidad pequeña como el que se venía usando anteriormente para los demás ambientes no diera buenos resultados, y se comportara de la misma manera que un clasificador aleatorio. La Figura 8.42 muestra la evolución de una red convolucional con la arquitectura del modelo de sanitario al ser entrenada con el *dataset* de salas.

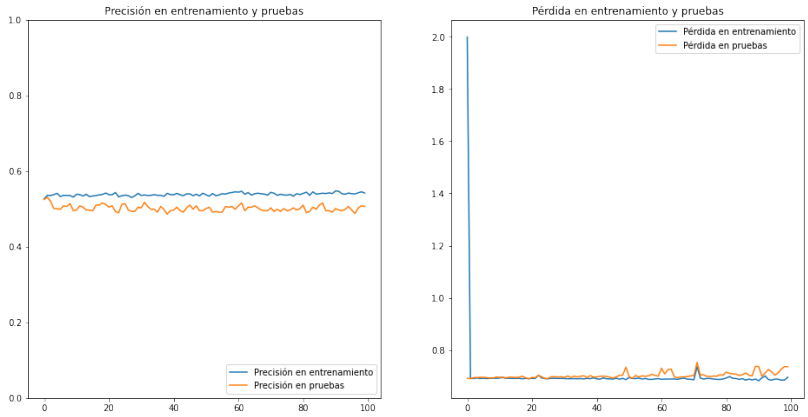


Figura 8.42: Evolución de efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de sala usando la misma arquitectura que en el modelo de sanitario.

Se detectó que parte del problema radicaba en que el optimizador escogido para entrenar todos los modelos (`Adam`) había encontrado un mínimo local en la función de pérdida al evaluar el conjunto de datos de entrenamiento de sala. Fue por esto que, únicamente en este modelo, se usó el algoritmo

de descenso del gradiente estocástico con un valor de taza de aprendizaje de 0.001.

Sin embargo, a pesar del cambio, el modelo seguía sin producir resultados significativamente mejores al clasificador aleatorio. Por esta razón fue necesario buscar referencias de modelos publicados cuyo punto fuerte fuera la clasificación de imágenes en un *dataset* complejo. Fue así como se decidió usar AlexNet [61] como referencia para el desarrollo de este modelo.

AlexNet es una red neuronal convolucional desarrollada en la universidad de Toronto por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton en 2012 para concursar en concurso ImageNet, del cual salió victoriosa. El artículo que la describe es considerado uno de los más influyentes en el medio de la visión computacional, y a la fecha ha sido citado más de 100,000 veces según Google Scholar. Consiste de 8 capas en total, las primeras 5 siendo convolucionales (algunas de ellas seguidas de capas de *max pooling*) y el resto siendo densas completamente conectadas. Si bien el problema que esta arquitectura quería solucionar era uno de clasificación multiclase, fue posible modificar su arquitectura inicial y entrenar un modelo similar con el conjunto de datos de entrenamiento y obtener un resultado de precisión cercano a 70 %. Este resultado es considerablemente mejor que el que se obtuvo con la arquitectura en los casos anteriores, como se puede notar en la Figura 8.43.

Figura 8.43: Evolución de efectividad y pérdida para datos de entrenamiento y pruebas en el modelo de clasificación binaria de ambientes de sala al usar una arquitectura basada en AlexNet y con un optimizador SGD con una taza de aprendizaje de 0.001.

Además, al probar el modelo con datos de validación y usando un umbral de 0.5, el modelo retornó resultados aceptables, como se puede notar en la Figura 8.44.

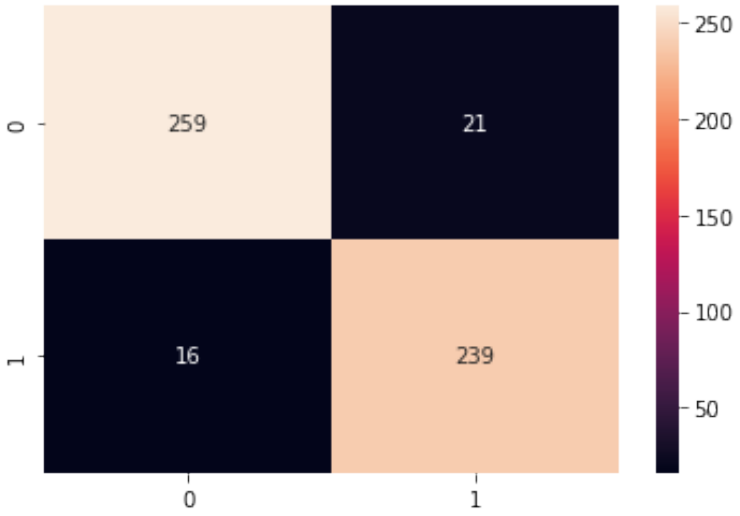


Figura 8.44: Matriz de confusión del modelo de sala al ser evaluado con imágenes de validación posterior al entrenamiento con un valor de umbral de 0.5.

Figura 8.45: Resultado del modelo de clasificación de comedor al recibir imágenes de salas reales.

Sin embargo, y a pesar de ser el modelo más robusto de todos los clasificadores binarios que se desarrollaron, éste también presenta limitaciones. En la Figura 8.45 se puede notar como, si bien clasifica correctamente la imagen de la sala (segunda imagen de la segunda fila), asigna un valor alto de certeza en la última imagen de la primera fila, que corresponde a una habitación. Este error en el modelo puede ser causado por el tamaño del conjunto de datos de entrenamiento (que para los demás modelos es suficiente, pero este en particular tiene una cantidad de parámetros mucho mayor) o bien por la concentración de elementos distractores en la imagen del dormitorio. Una manera de prevenir este problema sería incluir más datos y más variados en el *dataset* de entrenamiento, de manera que el modelo pueda aprender a clasificar ambientes con más ejemplos, y retocar la *augmentation layer*.

8.6.3. Implementación final

Para la implementación final del sistema de validación de imágenes se desarrolló un algoritmo en el cual, si una imagen desea ser ingresada al sistema, deberá ser procesada por cada uno de los mo-

delos de clasificación binarios y ser aceptada según su umbral de aceptación, resumido en la Tabla 8.1.

Ambiente	Umbral
Sala	0.5
Domitorio	0.5
Sanitario	0.9
Cocina	0.8
Comedor	0.85

Tabla 8.1: Ambientes y sus umbrales de aceptación

El acercamiento de tener varios modelos de clasificación binarios fue más efectivo porque favoreció la especialización de cada modelo en la clasificación de **su** clase, eliminando factores extra que pueden entorpecer el proceso de aprendizaje o desviarlo por completo hasta el punto en que el modelo no puede aprender correctamente. Además, aprovechó ventajas y fortalezas individuales de cada modelo sin comprometer ni afectar el comportamiento de sus compañeros. Es decir, en este acercamiento, cada modelo pudo dar una predicción de forma independiente y, si un modelo llega a presentar un comportamiento inesperado, dicho modelo puede ser atendido de manera independiente, el problema puede ser trazado y solucionado sin comprometer el funcionamiento de los demás. En otras palabras, este acercamiento provee trazabilidad.

Por otro lado, este acercamiento también facilita la escalabilidad. Si en un futuro se desea ampliar el alcance del proyecto para identificar también otro ambiente, solo será necesario entrenar un modelo que se encargue exclusivamente de ese nuevo ambiente y no se tendrá que modificar ninguno de los que actualmente funcionan. Esto no sucede con el acercamiento multiclase, en donde si el alcance del proyecto cambia, será necesario comenzar a entrenar un modelo nuevo desde cero.

Además, el acercamiento multiclase no fue capaz de dar una respuesta de rechazo consistente ante imágenes que no pertenecen a ninguna de las clases que tiene definidas. Esta es una consecuencia directa de que la capa de salida tenga una función de activación *softmax*, que a su vez es consecuencia del diseño mismo del modelo, que se planificó para que su salida fuera una distribución de probabilidad. En el caso que se deba rechazar una imagen, el modelo podría retornar un vector con todas sus entradas mostrando el mismo valor de $\frac{1}{n}$, en donde n es la cantidad de clases que se desean clasificar, o bien podría intentar reducir la entrada que *menos* se active con la imagen dada. Pero, como todas las posiciones del vector de salida dependen entre sí, minimizar una implica directamente aumentar el resto, por lo que incluso si una imagen no pertenezca a ninguna clase, podría ser clasificada como parte de una clase consistentemente. Este no es un problema en el acercamiento de modelos binarios, porque si una imagen no pertenece a ninguna de las clases, todas los modelos lo rechazarán y no habrán repercusiones de las decisiones de un modelo sobre otros.

Finalmente, el acercamiento de modelos binarios ayuda también al proceso de *Debuggeo*. Es decir, el proceso de encontrar la causa de un problema y solucionarlo en un sistema informático o de programación. Al ser modelos independientes, es fácil identificar si uno de ellos está comportándose de manera inesperada y, de ser así, únicamente se ha de trabajar en ese modelo. En el caso del acercamiento del modelo multicapa, si se observan comportamientos inusuales, será muy difícil, si no imposible, determinar qué componente está causando el error, y para solucionarlo se tendrá que entrenar toda la red neuronal completa desde cero.

8.7. Predicción de precios

Posterior al procedimiento de la matriz de correlación, el cual devolvió una matriz que contenía más de 140 columnas (Ver Anexo H), se seleccionaron las siguientes columnas por su alta correlación (influencia) en el precio de una propiedad según los datos con los que se contaba. Las columnas seleccionadas se encuentran listadas en la Tabla 8.2.

Columna	Descripción	Correlación con MARKETVAL
STORIES	No. de pisos en la construcción	0.1495
TOTROOMS	No. total de habitaciones	0.2474
BEDROOMS	No. de dormitorios	0.2004
BATHROOMS	No. de sanitarios	0.2620
FIREFPLACE	Chimenea usable	0.1951
DISHWASH	Lavadora de platos	0.1397
HINCP	Ingresos de la familia (*Houshold Income*)	0.3372
MORTAMT	Importe mensual total de la hipoteca	0.2263
more_3_bathrooms	Si tiene más de 3 sanitarios	0.2462
square_footage_3000_to_3999	Propiedad tiene entre 3000 y 3999 pies cuadrados de construcción	0.1245
square_footage_4000 or more	Propiedad tiene 4000 pies cuadrados o más de construcción	0.1963
hot_water_piped_gas	Propiedad tiene acceso a agua caliente por cilindro de gas	0.1339

Tabla 8.2: Columnas correlacionadas con MARKETVAL.

Como se puede notar en la Figura 7.22, los valores de precio se encuentran concentrados en el rango $[0, 10^6]$. También la Tabla 7.1 indica que más del 75 % de los valores de *MARKETVAL* se encontraban dentro de este rango. Incluir la totalidad de los datos sabiendo que el valor que se busca predecir presentaba un sesgo tan fuerte hacia el rango descrito introduciría ruido al modelo en el entrenamiento y afectaría negativamente el rendimiento en general del sistema. Por este motivo se decidió excluir todos los registros cuyo valor de *MARKETVAL* fuera superior a $\$1,000,000$.

Después de excluir los datos mayores a $\$1,000,000$, el conjunto se describía de la siguiente manera:

No. de elementos	31349
Media	298202.35
Desv. estándar	210775.06
min.	1000.00
25 %	144075.00
50 %	250481.00
75 %	403872.00
max.	999836.00

Tabla 8.3: MARKETVAL luego de excluir valores mayores a $1 millón.

8.7.1. Modelo de *Random Forest*

Como primera instancia se entrenó un modelo de regresión basado en el algoritmo de *random forest* con los parámetros detallados en el siguiente bloque de código. El conjunto de datos se separó en un 80 % de entrenamiento y 20 % de validación.

```
rf = RandomForestRegressor(
    n_estimators = 10000,   # 10k arboles en el bosque
    random_state = 42,
    max_features='log2',    # log2 de la cantidad de features totales
    max_depth=10,           # Profundidad maxima de cada arbol. Fijada en 10 para
    mantener un proceso ligero
    n_jobs=-1               # -1 representa que se distribuira la carga en todos los
    nucleos de procesamiento disponibles
  )
```

Cabe mencionar que la librería se encarga de implementar algoritmos de embolsado por defecto, por lo que no es necesario que se le especifique en ningún momento de forma explícita.

Como métricas de rendimiento del modelo se usaron el error medio absoluto (EMA) y la desviación estándar del vector de resultados.

Se obtuvo un EMA de $118,021.62 con una desviación estándar de $110,691.03.

8.7.2. Modelo usando *XGBoostRegressor*

Luego de que los resultados del *random forest* básico no fueran los mejores, se decidió intentar con una versión modificada del mismo, usando también *XGBoost* a través de la librería `xgboost` de Python3.

```
model = XGBRegressor(
    n_estimators=1000,  # 1000 arboles en el bosque
     max_depth=10,      # Profundidad maxima de cada arbol. Fijada en 10 para
    mantener un proceso ligero
     eta=0.3,           # Para evitar sobre ajustes
     subsample=0.85,    # Proporcion de submuestra de las instancias de entrenamiento
     colsample_bytree=0.45,     # Relacion de submuestra de las columnas al construir
     cada arbol.
)
```

Luego de entrenar el modelo se obtuvo una desviación estándar de $3,546.59 y un EMA de $133,554.73. Este modelo presentó un valor de desviación estándar 95 % menor que el modelo anterior, lo cual indica que los resultados se encontraban con menor dispersión y, por tanto, podían ser usados con mayor seguridad por los usuarios del sistema. Sin embargo, también presentó un valor EMA 45 % mayor.

Una de las principales razones por las cuales se decidió usar esta librería fue porque permite, a través de su función `RandomizedSearchCV`, ejecutar una búsqueda automática de un conjunto de hiperparámetros que minimizaran el valor de pérdida del modelo. El siguiente bloque de código muestra el conjunto de hiperparámetros a probar.

```
n_estimators = [int(x) for x in np.linspace(start = 500, stop = 1000, num = 30)]
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)
random_grid = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'tree_method': ['gpu_hist'],    # Para que corra el entrenamiento en GPU
    'eval_metric': ['rmse']
}
```

Sin embargo, se descubrió que el conjunto de datos no era apto para este tipo de entrenamientos, pues a pesar de que Google Collab proporcionaba ambientes de alta RAM, ninguno fue suficiente para todo el procesamiento que necesitaba un conjunto de datos tan grande como el que se tenía, y al intentar con muestras de menor tamaño, los resultados no mantenían el rendimiento esperado al trasladarse al conjunto de datos completo.

Por este motivo se decidió que el modelo final fuera el de *XGBRegressor*, pues es el que minimiza la desviación estándar de los resultados del conjunto de datos de evaluación, resultando en un intervalo de confianza menos disperso. Si bien es cierto que su error medio absoluto medio es considerablemente mayor que su contraparte *random forest* simple, no supera el 15 % del valor máximo a considerar ($1 millón), por lo que aún puede considerarse aceptable.

Cabe mencionar que, al hacer pruebas con datos que propiedades cuyo valor real en mercado supera el límite de $1 millón que se estableció para el análisis, se encontró que el EMA tenía un valor de $1,231,860.17, con una desviación estándar de $1,303,621.29. Si bien el valor de la desviación estándar es considerablemente mayor, puede explicarse al tomar en cuenta que el rango de precios evaluados está contenido en el rango $[10^6, 10 * 10^6]$.

Finalmente, en el producto final, el modelo toma como parámetros de entrada los valores que se listan en la Tabla 8.2 y los usará para computar un precio sugerido P. Luego, tomando en cuenta la desviación estándar σ del modelo se predice un rango de precios aceptables

$$[P - \sigma, P + \sigma]$$

.

8.8. API *web*

Ante el problema de exponer un servicio a internet surgen una infinidad de posibles soluciones, cada una con sus particularidades y sus pros/contras. En este caso, que el servicio en cuestión utilizaba algoritmos de alta exigencia para el hardware del servidor en que esté corriendo, la premisa fue **no cargar al servidor más de lo que es estrictamente necesario**. Además, una preocupación igual de importante fue la de usar un *framework* que sea capaz de responder y escalar el uso de recursos del servidor cuando tenga que manejar solicitudes en concurrencia.

Es por esto que se decidió usar Flask como *framework* principal en el desarrollo del servidor *web* que expondrá los servicios de clasificación de imágenes y predicción de precios. Dado que Flask es 100 % compatible con WSGI, su configuración con es directa y no necesita de mucha configuración extra. Además, a través de esto es posible hacer que el *backend* aproveche las capacidades de GUnicorn de configurar ya sea una cantidad especifica de hilos de procesamiento para manejar solicitudes, o incluso configurar la cantidad de procesos que el servidor HTTP tendrá a su disponibilidad en el

momento que los comience a necesitar.

Incluso, si habilitar una cantidad de procesos disponibles mayor no funcionara y el API presentara *delays* o colas muy largas, es posible configurar el *backend* para que coleccione solicitudes por un tiempo determinado cuando está bajo estrés (este tiempo puede ser 500ms o menos, dependiendo del momento en que se encuentre) y unir las entradas en un solo *batch* para que el modelo de aprendizaje lo procese en un solo hilo, sin necesidad de utilizar un hilo por cada solicitud. [56]

Por otro lado, se decidió seguir una arquitectura REST para el diseño del API pensando en que el servidor no debe ejecutar nada que no sea estrictamente necesario. *Frameworks* como Flask, montados sobre Python y corriendo en conjunto con WSGI y GUnicorn son suficientemente ligeros como para no representar una carga extra para el hardware que tenga que procesar las solicitudes con modelos de IA o árboles de decisión. Este no es el caso con el **protocolo** SOAP, el cual si bien es cierto que garantiza principios ACID, estabilidad y escalabilidad, lo hace al precio de ocupar más tiempo en el procesador, y en este caso, se trata de tiempo que no se está aprovechando para calcular una respuesta a una solicitud.

Además, la arquitectura REST es, actualmente, mucho más usada que cualquier otra en servidores *web* [1] , y al ser emparejado con respuestas en formato JSON, garantiza no solo que la información sería legible por cualquier otro *software*, sino que por cualquier otro humano que pueda obtener una respuesta del servidor.

8.9. Controles de acceso

Como se explicó anteriormente se implementó el control de acceso por correo o *wallet* y contraseña, así como validaciones al momento de registrarse para hacer más segura la plataforma. En las siguientes imágenes se puede ver esta implementación en la plataforma.

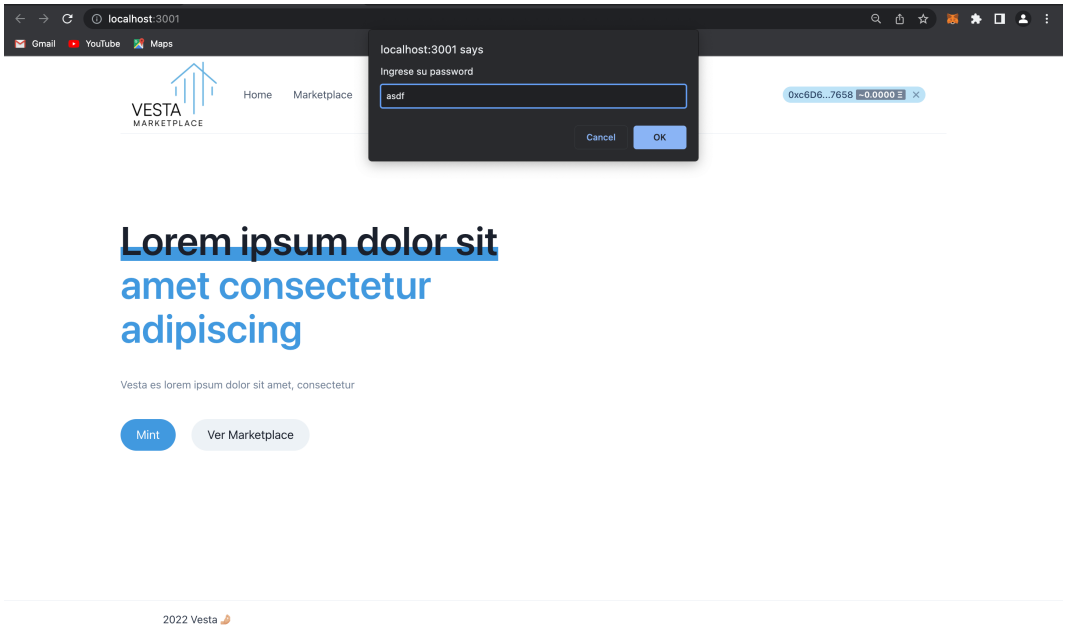


Figura 8.46: Registro de usuario

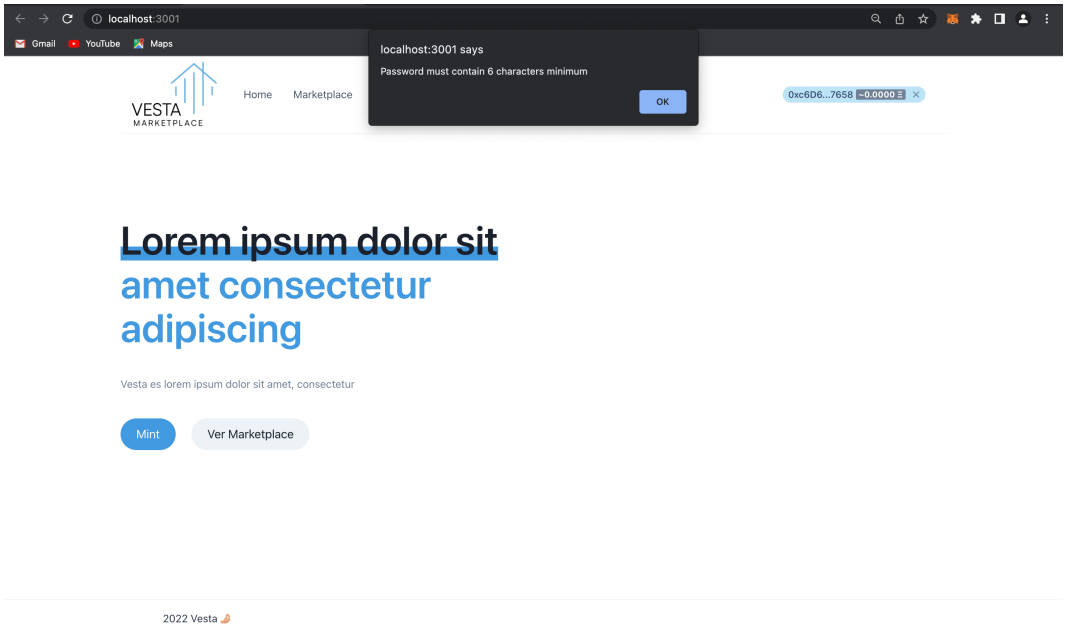


Figura 8.47: Contraseña insegura

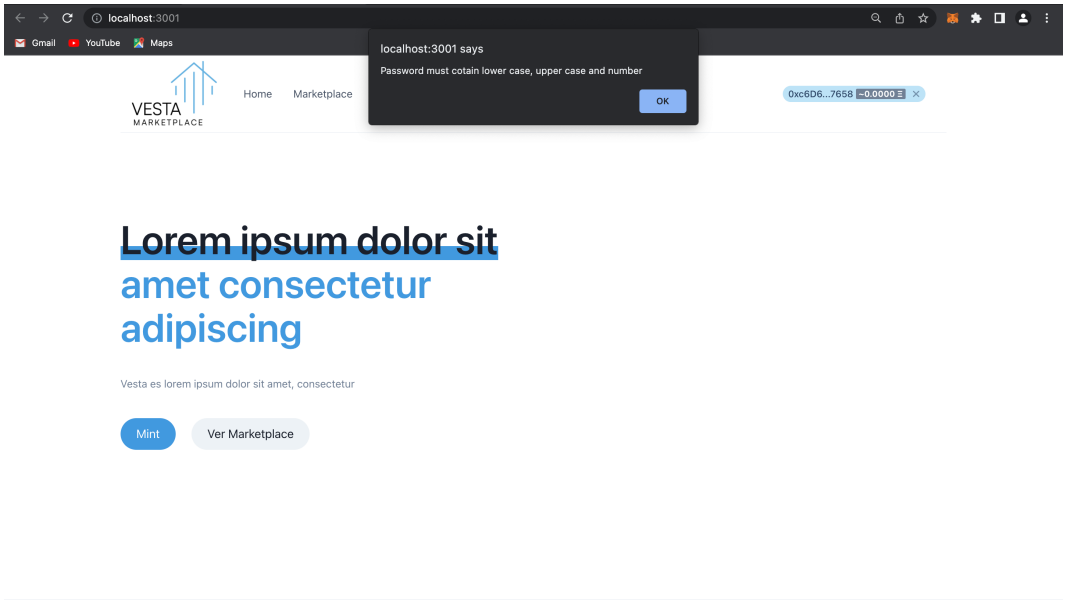


Figura 8.48: Contraseña insegura

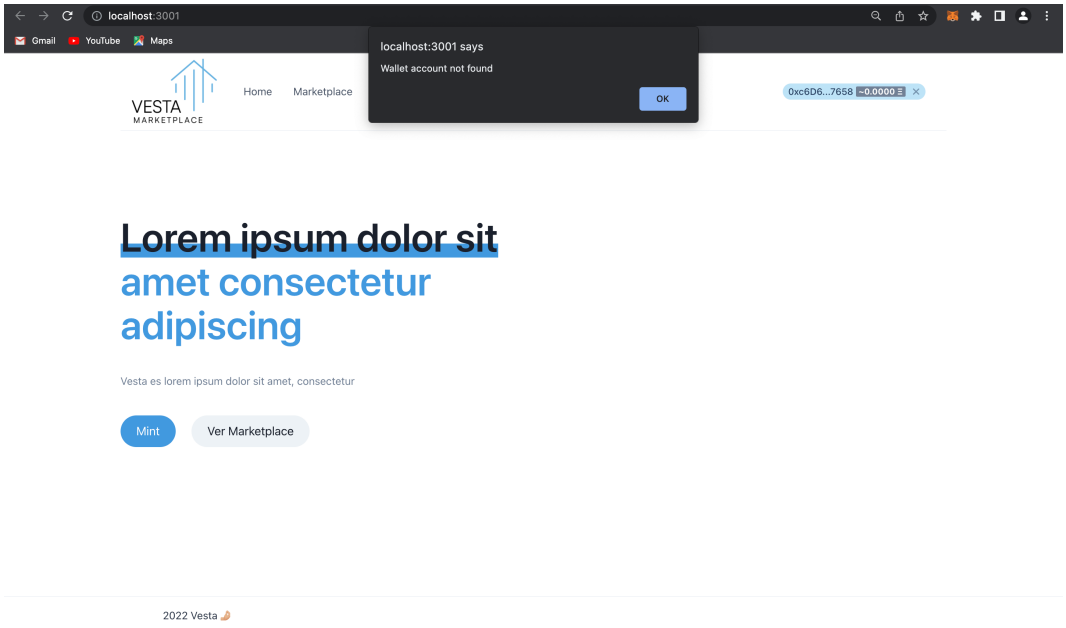


Figura 8.49: Inicio de sesión fallido

Como se puede observar en las figuras, la plataforma no permite el ingreso de una contraseña que no cumpla con los requisitos, así como si no hay una cuenta no permite el acceso. Esto nos ayuda a mantener más segura la plataforma. Es más difícil que lleguen a obtener una contraseña a través de algún ataque, ya que hay mas opciones de combinaciones de caracteres. Así como para conectarse por medio de la *wallet* necesitarían saber también su contraseña y frase de la *wallet* para poder conectarla. Es importante que la seguridad en los controles de acceso sea eficiente, ya que si

alguien consigue entrar en una cuenta, esta persona podría publicar alguna propiedad o comprar una propiedad si tiene su *wallet* conectada. Esto puede implicar grandes cantidades de dinero. En la sección de recomendaciones se sugieren otras maneras de mejorar los controles de acceso.

8.10. KYC

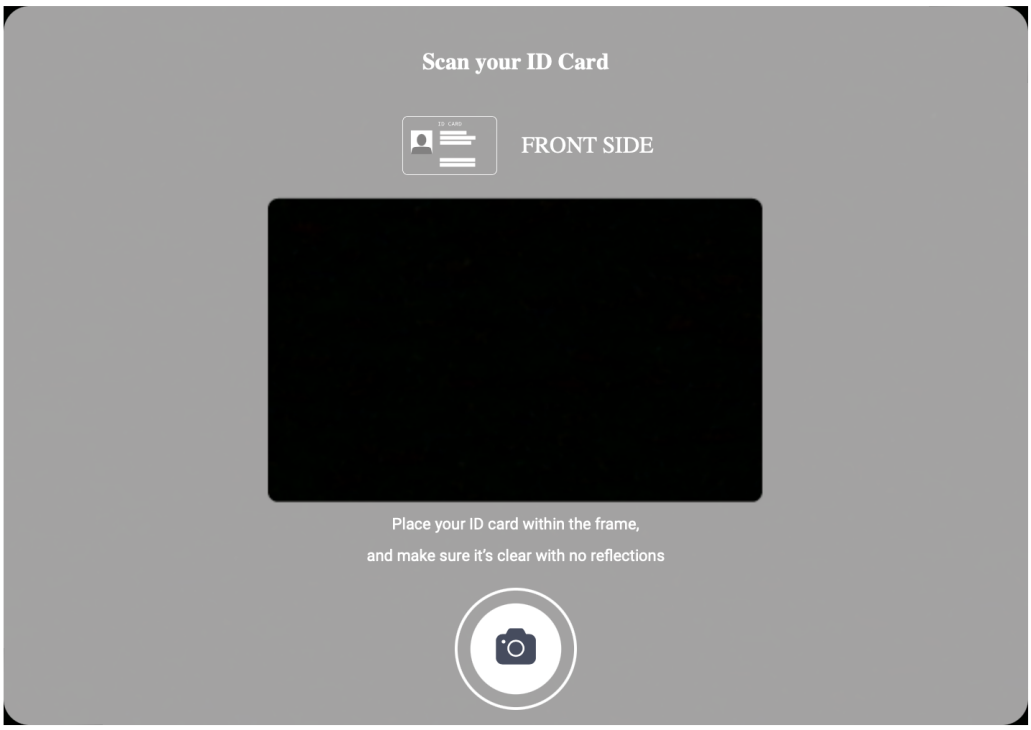


Figura 8.50: Subir foto de documento de frente para autenticación con KYC

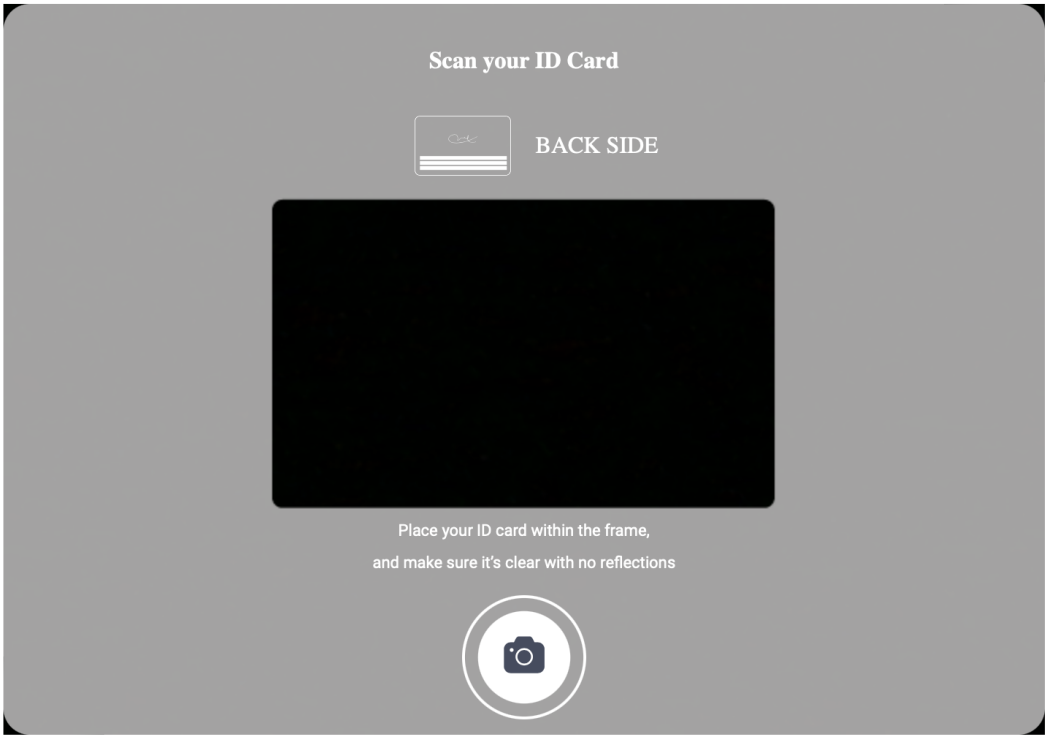


Figura 8.51: Subir foto de documento por atrás para autenticación con KYC

Figura 8.52: Subir foto de rostro para autenticación con KYC

En las imágenes se puede ver el proceso para la autenticación con KYC, como se puede ver se solicita algo que el usuario tiene, en este caso el documento de identificación. Así como se solicita algo que el usuario es, una foto de su rostro. Estos los archivos los verifica la plataforma para validar que tanto el documento como la foto coincidan con la misma persona. Esto nos ayuda a garantizar que la persona sea la que dice ser y así poder evitar estafas, ya que esta verificación se debe hacer antes de subir una propiedad o de realizar una compra.

8.11. JSON *web token* y cifrado de información

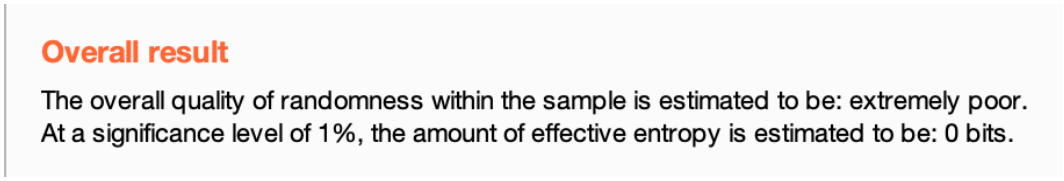


Figura 8.53: Resultados entropía con algoritmo RSA

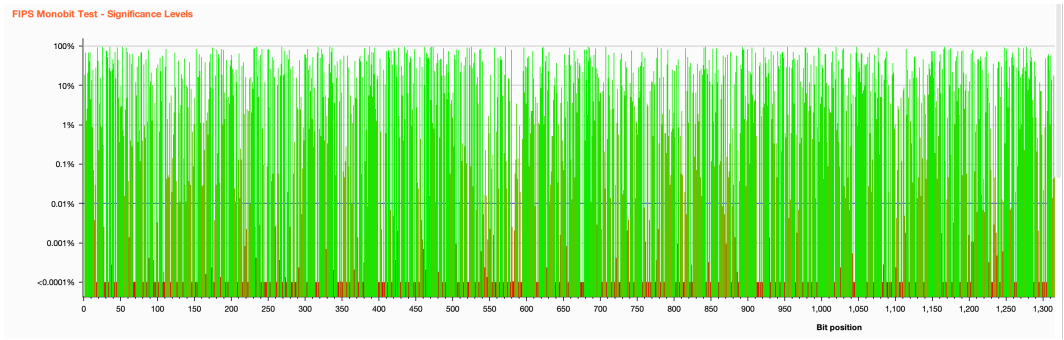


Figura 8.54: Monobit *test* con algoritmo RSA de 0 hasta 1300 bits

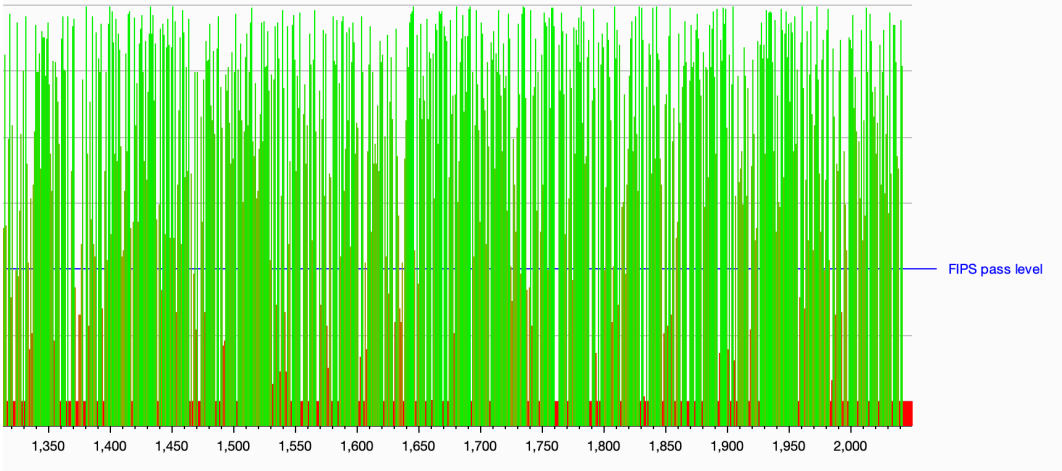


Figura 8.55: Monobit *test* con algoritmo RSA de 1350 hasta 2000 bits

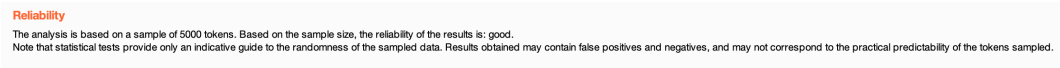


Figura 8.56: Fiabilidad con algoritmo RSA

Para los JWT implementados con RSA se decidió calcular su entropía para ver que tan aleatorio está siendo la creación de estos y así sea menos probable que un atacante pueda obtener uno de estos tokens y *crackee* la plataforma.

En la Figura 6.5 se puede ver en general los resultados de la entropía del token con el algoritmo RSA fueron extremadamente pobres. En las siguientes dos figuras (6.6 y 6.7) podemos ver el *monobit* test que se realizó con los tokens. Este *test* consiste en ver para cada uno de los *bits* si la cantidad de veces que el *bit* tiene el valor 0 o 1 coincide con la cantidad de veces esperada en una secuencia totalmente aleatoria. Por último, en la Figura 6.8 tenemos la fiabilidad de los tokens considerando que pudieron existir falsos positivos o negativos.

A simple vista esos resultados pueden parecer algo confusos o contradictorios, ya que como va a ser que la entropía salió muy baja, pero aún así sean fiables. Esto se puede deber a la estructura por la que están conformados los tokens. Un JWT está conformado por tres partes. Primero el *header*, el cual tiene el algoritmo que se utilizará y el tipo de token. Luego el *payload* aquí es donde esta la información que se quiere guardar en el token. La tercera es la firma, la cual nos ayuda a verificar después si la integridad del token ha sido comprometida en algún momento. Tanto el *header* como el *payload* son cifrados con base64, entonces para estas dos partes la llave privada no tiene un uso. Para estas pruebas se utilizó tanto el mismo *header* como el mismo *payload*, por lo que al cifrar estas partes con base64 siempre dio el mismo resultado, esto hace que gran parte del token ya no sea del todo aleatorio porque dos partes del token son iguales siempre.

Sin embargo, si observamos las gráficas anteriores vemos que para la mayoría de los *bits* si tienen un nivel de aleatoriedad esperado, que son los que están en verde, y solo algunos no dieron la cantidad esperada. Al ser el algoritmo RSA la seguridad de la está bastante ligada con la cantidad de *bits* de la llave en este caso al tener bastantes *bits* y que la mayoría si tienen un nivel esperado, podemos

ver en la Figura 8.8 que por eso es que los tokens son fiables.

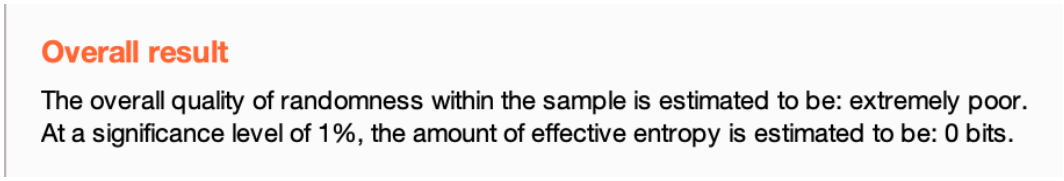


Figura 8.57: Resultados entropía con algoritmo HMAC

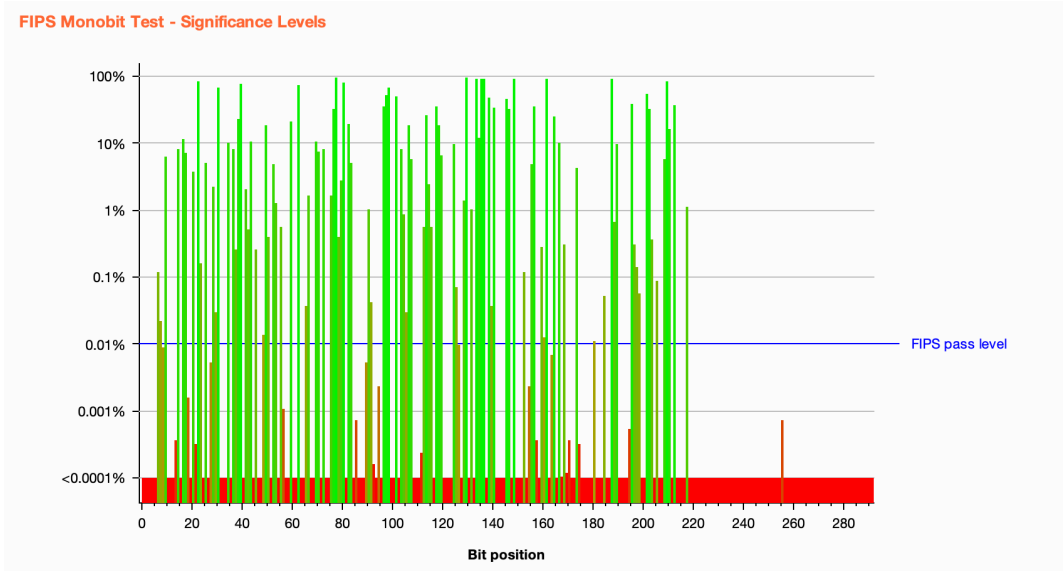


Figura 8.58: Monobit *test* con algoritmo HMAC

Figura 8.59: Fiabilidad con algoritmo HMAC

También se realizaron las mismas pruebas para los tokens, pero usando el algoritmo de HMAC, esto para poder comparar con los de RSA y ver si sí había más seguridad usando RSA. En la figura 6.9 podemos ver que al igual que con RSA la entropía fue bastante baja. Al igual que con RSA el *header* y *payload* eran iguales para todas las peticiones de prueba que se hicieron, por lo que también tenían una parte del token igual. En la figura 6.10 se ve la gráfica del Monobit *test*, aquí ya se ven diferencias, la principal es que la gráfica es bastante más pequeña esto es porque las llaves con HMAC no son de un tamaño tan grande como las de RSA, por lo que hay menos *bits* que analizar. También se puede ver que hay varios *bits* con un nivel de aleatoriedad bastante bajo, sobretodo los del final. Esto hace que la fiabilidad con estos tokens sea razonable, como se ve en la Figura 6.11, y no buena como con los tokens con RSA.

Existe el riesgo de que logran obtener un token, al no tener una entropía muy buena. Esto implicaría que el atacante podría realizar cualquier petición al servidor, ya que el sistema pensaría

que si inició sesión y es el usuario correcto usando la cuenta. Si analizamos cual podría ser el resigo máximo, sería si la cuenta que está usando tiene alguna *wallet* vinculada, ya que de esta forma podría comprar alguna propiedad fácilmente o modificar alguna que haya publicado ese usuario y hacerle perder grandes cantidades de dinero.

8.12. Análisis de riesgos y políticas de seguridad

8.12.1. Análisis de riesgos

Riesgo	Impacto	Contramedida
Información comprometida	Si se compromete nuestra información esta puede llegar a ser modificada, borrada o robada. Esto implicaría pueden modificar datos importantes de las propiedades, borrar propiedades que publicaron los usuarios u obtener data relevante sobre los usuarios y sus propiedades	Implementar *audits* para saber la hora a la que fuera borrada y por cuál usuario, así como mejorar la seguridad de los controles de acceso y cambiar las llaves privadas des los usuarios.
Llaves privadas comprometidas	Pueden descifrar la información que tengamos guardada que se encuentre cifrada, dando acceso a esta información confidencial. Así como poder obtener un token JWT y hacer cualquier petición que quieran al servidor.	Cambiar las llaves privadas a todos los usuarios, enseñar donde son los lugares correctos en donde guardar las llaves.
Acceso no autorizado a los recursos	Podría obtener, borrar o modificar información que no debería de poder ver el usuario, ya que no cuenta con los permisos correspondientes. Navegar y realizar acciones por partes de la plataforma que no debería poder ver comprometiendo la confidencialidad.	Mejorar los controles de acceso, analizarlos para buscar posibles vulnerabilidades por las cuales se pudo haber accedido.
Caída del sistema	Los diferentes recursos de la plataforma no estarían disponibles para los usuarios, afectando las posibles ventas y transacciones que se estaban realizando en ese momento y evitando que se puedan ver las propiedades en venta.	Tener planes de respaldo en caso de caídas del sistema y que no pase mucho tiempo desde que se cayó hasta que volvió a funcionar. Contar con información de respaldo para que no se pierda la información que se tenía guardada. Evitar que se haga cualquier transacción si se cae el sistema.

Riesgo	Impacto	Contramedida
Phishing	Podrían engañar a nuestros usuarios enviándoles correo haciéndoles pensar que son de parte de nuestra plataforma y arriesgándolos a comprometer su información personal.	Hacerle saber al usuario de que nunca se les pedirá sus datos privados y hacerles saber cómo se ve un correo de la plataforma.
Propiedades falsas publicadas	Los usuarios pueden comprar una propiedad la cual no es la que están viendo y ser estafados. Haciéndoles perder confianza en la plataforma.	Verificar cada vez que se suba una nueva propiedad que de verdad represente una propiedad y verificar que los documentos sean legales.
JWT comprometido	Pueden realizar cualquier petición sin importar si tienen usuario o no, ya que se hacen pasar por otro usuario. Pueden obtener acceso a recurso que no deberían poder ver. Podrían modificar o borrar información sensible.	Asegurar la seguridad de los JWT, verificar el algoritmo que se está usando y ver si se puede mejorar, cambiar las llaves privadas.
Vulnerabilidades en el *smart contract*	Podría comprometer la seguridad de las transacciones y el traspaso de los *NFTs* explotando esta vulnerabilidad.	Asegurar que la programación del *smart contract* sea correcto, probándolo en una red de prueba
No establecer tarifas límite en bloques	Pueden haber transacciones que se cobre demasiado o que se cobre muy poco a lo que debería ser. Esto hace que los usuarios ya no quieran comprar o perdamos dinero	Establecer una tarifa mínima y máxima de límite de gas para las diferentes transacciones
SQL *Injection*	Poder obtener información y ejecutar acciones en la base de datos. Afectando la información al poder modificar o borrar datos. Comprometiendo la confidencialidad del sistema y los datos de los usuarios.	Implementar medidas de seguridad contra SQL *Injection* como limpiar los *queries* que se realizan.

Como se puede ver en el análisis de riesgos anterior, se identificaron varios riesgos en diferentes áreas de la plataforma. Estos son posibles riesgos hacia nuestra plataforma cada uno tiene una probabilidad diferente de que ocurra, pero aún así debe considerarse y pensar cómo puede afectarnos.

8.12.2. Políticas de seguridad

- Política de cifrado aceptable: Se pueden usar algoritmos de cifrado los cuales ya se han demostrado que funcionan y han recibido varias revisiones públicas.
- Política de respuesta a la violación de datos: Se debe definir qué hacer y las personas encargadas en realizar las diferentes acción al momento que haya una violación de datos.
- Política de credenciales de la base de datos: Se debe definir en dónde y cómo se guardan las credenciales necesarias para la base de datos de la plataforma.

- Política de aceptación de firma digital: Definir los requisitos para cuando una firma digital se considera un medio aceptado para validar la identidad.
- Política del Plan de Recuperación de Desastres: Definir e implementar un plan de recuperación ante desastres que describa el proceso para recuperar los sistemas de TI, las aplicaciones y los datos de cualquier tipo de desastre que provoque una interrupción importante.
- Política de correo electrónico: Define cómo se debe usar el correo electrónico así como los casos de uso en donde es aceptable y no el uso del correo.
- Plan de protección de clave de cifrado: Definir los requisitos para proteger las claves de cifrado.
- Política para la construcción de contraseñas: Definir las pautas y mejores prácticas para la creación de contraseñas seguras.
- Política de protección de contraseña: Definir el estándar para la creación de contraseñas seguras, la protección de esas contraseñas y la frecuencia de cambio.
- Política de evaluación de riesgos: Definir que se pueden realizar evaluaciones periódicas de análisis de riesgos con el fin de determinar áreas de vulnerabilidad e iniciar la prevención adecuada.
- Política del plan de respuesta de seguridad: Definir el plan de respuesta de seguridad a seguir para que se pueda formal una respuesta exitosa en caso de que ocurra un incidente.
- Política de auditoría del servidor: Definir los estándares de configuración para los servidores instalados.
- Política de seguridad del servidor: Definir estándares para la configuración de seguridad para servidores.
- Política de concientización sobre ingeniería social: Definir pautas para generar conciencia sobre la amenaza de la ingeniería social y define procedimientos cuando se trata de amenazas de ingeniería social.

Para poder reducir lo máximo posible el riesgo que existe a que una amenaza suceda, se listaron las políticas de seguridad anteriores las cuales al definirlas y cumplirlas ayudarán a tener una mejor organización y control de la seguridad. Además, ayuda a que todos los integrantes del equipo sepan lo que deben hacer y en qué momento. Es importante que se cumplan estas políticas o la seguridad de la plataforma podría llegar a verse comprometida.

8.13. Validación legal

Basado en la entrevista con el Oficial Mayor de la Corte Suprema de Justicia, en cuanto al traspaso inmediato del título de propiedad, siempre es requerido el debido proceso notarial de la escritura pública. Para que el *NFT* tenga y represente el valor legal de los derechos sobre la propiedad, es requerido una modernización de la ley. Se debe actualizar la ley para que incluya aspectos sobre las tecnologías y procesos tecnológicos digitales actuales. Sin embargo, este derecho sobre la propiedad puede ser aplicable inmediatamente a derechos sobre bienes muebles y títulos de crédito. En cuanto a los bienes inmuebles, el *NFT* podría reemplazar la razón de inscripción electrónica del Registro de la Propiedad, que equivaldría a la transacción del traspaso del título de propiedad, siempre y cuando se haga la debida referencia en el blockchain hacia la escritura pública; almacenando un correlativo y campos adicionales que identifiquen la propiedad.

Basado en la investigación de los entes involucrados como el Registro de la Propiedad y la SAT, y gracias a que en la actualidad estos dos tienen un acuerdo de cooperación, el sistema podría ayudar directamente al Registro de la Propiedad para verificar cualquier transacción de títulos realizada, mientras que la SAT podría de igual manera, realizar el cálculo de impuestos relacionados a los bienes inmuebles ya que todos los usuarios de la plataforma deben estar debidamente identificados.

CAPÍTULO 9

Conclusiones

- Se desarrolló un *smart contract* capaz de producir *NFTs* con metadatos personalizados que refleja las especificaciones de un inmueble.
- Se desarrolló un *smart contract* transparente de tal forma que toda transacción que lo involucra es plasmada en todos los exploradores de bloques de la red de Ethereum
- Se desarrollaron *NFTs* bajo lo estándares de desarrollo de tal manera que todo *NFT* creado por el *smart contract* puede ser visualizado en cualquier sitio web de *NFTs* como OpenSea y Rarible.
- Se implementó la transferencia de *NFTs* entre billeteras digitales, haciendo posible el traspaso de la propiedad de un inmueble tokenizado.
- Los metadatos del *NFT* es almacenada dentro de *blockchain* por la necesidad de inmutabilidad de datos, sin embargo, por temas de rendimiento y limitaciones de la red, la imagen representativa es almacenada fuera de *blockchain*.
- *Web3* ayuda a interactuar fácilmente con un *smart contract* de un red *blockchain* como si se tratase de realizar consultas a un API tradicional.
- Se logró crear un *marketplace* basado en *blockchain* que automatiza el proceso de transferencia y pertenencia de derechos de propiedad, tokenizando las propiedades en la red. Esto implica que se puede aprovechar las características que nos provee *blockchain* como trazabilidad, confiabilidad, seguridad y pertenencia; e integrar con aplicaciones centralizadas cuando se desea llevar un control sobre las identidades y entidades del sistema.
- Se diseñó e implementó una infraestructura segura en la nube. Esta soporta un servidor web que contiene el *backend*. Cuando se integra un *backend* centralizado con *blockchain*, el soporte de concurrencia de la infraestructura debe ser directamente proporcional al de la concurrencia en la red *blockchain*.
- Se diseñó e implementó una base de datos y *backend* que provee la API que la aplicación emplea para identificar usuarios en la red de *blockchain*. De este modo se elimina el anonimato de la red. Además, el *backend* proporciona información adicional que no se almacena en el *blockchain* y es de utilidad en el *frontend* para mostrar datos al usuario. La integración de *backend* centralizado y *blockchain* nos permite reducir la cantidad de información que se almacena en la red, lo cual implica reducción de costos transaccionales y de almacenamiento para cada nodo participante.

- Se creó un oráculo que sirve como fuente de información entre el *blockchain* y los datos externos al *blockchain*. Estos nos sirven para compartir información y proporcionarla a la red.
- Según lo experimentado en la elaboración de este proyecto, lo mínimo requerido para integrar *blockchain* con un *backend* es proporcionar en la base de datos una tabla adicional donde se pueda almacenar las *wallets* que los usuarios pueden tener, y agregar un campo de identificación en las entidades que también se almacenan en la red para poder relacionarlas. Una aplicación basada en *blockchain* puede funcionar fácilmente sin un *backend* tradicional centralizado, porque la red de *blockchain* es el equivalente a la base de datos y los *smart contracts* que almacena actúan como *backend* proporcionando un API para su uso; de este modo el *frontend* puede hacer uso de *Web3* para consultar los *smart contracts* usándolos como API para poder acceder a la información almacenada en la red.
- La implementación de CI que ejecute *tests* y otras validaciones, simplifica el ciclo de vida de programación y permite la identificación de errores con anticipación en diferentes etapas del ciclo de vida de la programación. La elaboración de *tests* automatizados implica un costo de tiempo mayor en la elaboración de un requerimiento, sin embargo reduce significativamente las probabilidades de errores finales. El usar adecuadamente las herramientas, *frameworks*, aplicaciones y protocolos reducimos significativamente las vulnerabilidades que ingresamos al sistema indirectamente al momento de programar y configurar.
- El acercamiento de usar varios modelos de redes neuronales convolucionales de clasificación binaria presentó mejores resultados que su contraparte multiclase para la clasificación de imágenes en clases de ambientes de una casa.
- Se desarrolló un sistema de verificación de datos usando imágenes de ambientes de una casa fue alcanzado con el acercamiento de tener 5 modelos de clasificación binaria independientes.
- El ambiente más fácil de reconocer por los modelos es del sanitario, principalmente porque contiene una menor cantidad de elementos distractores en la imagen. Esto también lo evidencian los resultados, que indican que este ambiente tuvo cerca de 75 % de efectividad.
- El algoritmo de *Random Forest* simple fue superado por su variante usando *XGBoost* para optimización de hiperparámetros.
- Se desarrolló un sistema de recomendaciones de precios para potenciales clientes de propiedades fue alcanzado con el modelo de *XGBRegressor*.
- Se desarrolló un API web usando Python y Flask como *framework*, siguiendo una arquitectura REST para aligerar la carga en la instancia del servidor y, al mismo tiempo, darle la flexibilidad necesaria para escalar cuando la situación lo requiera.
- Se definieron las políticas a seguir, en base a las necesidades y requerimientos de nuestra plataforma, para mejorar la seguridad de esta.
- Por medio de la implementación de el inicio de sesión con contraseña y KYC se garantizó la seguridad de la información de los usuarios en la plataforma.
- Se garantizó la confidencialidad, disponibilidad e integración de los datos con la implementación de el algoritmo de cifrado RSA, así como la transferencia de datos segura por medio de JWT.
- Por medio de investigación, discusión y trabajo de campo se detectaron varias posibles amenazas, así como el impacto que tendrían y cómo se pueden resolver en caso se presenten.

CAPÍTULO 10

Recomendaciones

Smart contract Encontrar una manera de que el usuario no interactué dos veces con el contrato cuando se cree un *NFT*, enviando los metadatos directamente en el *mint*, para evitar el doble pago de comisión de la red.

- Implementar un estándar, como el ERC-994, como estándar del *smart contract* principal ya que este es un estándar relativamente nuevo pero específicamente para el registro de tierras y propiedades físicas.
- Implementar *NFT burn* para eliminar *NFTs* en circulación para casos de uso como remodelación de un inmueble que ya se encuentra tokenizado, de tal manera que elimine su actual propiedad y cree uno nuevo.
- Almacenar más campos relevantes en el *smart contract* como el número de Finca, Folio y Libro, así como el relativo a la escritura pública para que el proyecto tenga una validación legal fortalecida.

Web3 Implementar IPFS para que el usuario cargue una imagen y esta se despliegue en la red de IPFS y sea esta la representativa del *NFT*.

- Producir un documento PDF a la hora que se realiza un *mint* y una transferencia como respaldo de la transacción.

Database Se recomienda modificar que datos son almacenados en la base de datos para cubrir diferentes casos de uso como lo es el registro de propiedad en otros países y los datos relevantes de otros tipos de propiedad como vehículos o propiedad intelectual.

CD Se recomienda implementar un flujo de CD (*Deployment* Continuo) para completar el ciclo de vida de programación automática.

Test Se recomienda cubrir lo más cercano al 100 % de cobertura de código con *tests* de todos los sistemas utilizados *backend*, *frontend* y adicionales como los *test* de *smart contracts*.

Alcance Este proyecto se puede generalizar para utilizarse en otros ámbitos como un registro de propiedad de otros bienes como vehículos, identificaciones personales y otros.

Seguridad Agregar un módulo de auditoría para poder llevar registro de las acciones de todos los usuarios en caso haya algún incidente.

- Se recomienda agregar librerías o sistemas de identificación personal para el usuario.

Dataset Para el desarrollo de modelos de clasificación binaria de ambientes de una casa se recomienda encontrar un *dataset* con una mayor cantidad de datos (al menos 50 mil), o bien darse a la tarea de armar un *dataset* nuevo con imágenes del mundo real al alcance.

- Se recomienda usar un *dataset* con más ambientes o bien recopilar y etiquetar imágenes que no se limiten únicamente a ambientes de una casa, sino que puedan incluir imágenes variadas de todo tipo con el propósito de poder entrenar a los modelos binarios con ejemplos de imágenes que no deben ser aceptadas, y que tampoco deban ser aceptadas por otros modelos.

Modelos IA Uno de los factores de mayor influencia en las debilidades de los modelos desarrollados fue la presencia de elementos distractores el fondo de las imágenes utilizadas. Se recomienda desarrollar un modelo o algoritmo que sea capaz de reconocer objetos o elementos con cualquier tipo de fondo, logrando separarlo y usar eso como entrada para un modelo más complejo de clasificación.

- Una limitante importante en el desarrollo del modelo de sugerencia de precios fue la capacidad de la máquina *host*, que fue incapaz de procesar en su totalidad una búsqueda aleatoria de hiperparámetros para optimizar un modelo de tipo `XGBRegressor`. Se sugiere desarrollar una solución que permita ejecutar este tipo de algoritmos con grandes volúmenes de datos de una manera más eficiente en cuando a uso de memoria volátil.

- El algoritmo que permite el escalamiento de poder de procesamiento a nivel de instancia de servidor depende de configuraciones de librerías como Tensorflow que no son accesibles por el desarrollador de manera directa. Se recomienda implementar un algoritmo de escalamiento personalizado que pueda mejorar el rendimiento genérico de las librerías para este caso particular.

Análisis de riesgo Conforme la plataforma vaya creciendo, cada vez es más propensa a más amenazas por lo que este análisis y las políticas deben ser revisadas y actualizadas cada cierto tiempo no muy lejano, para que siempre todas las personas que trabajan en la plataforma estén al tanto y pendientes de cualquier vulnerabilidad.

Controles de acceso Implementar maneras adicionales para manejar la autenticación por KYC, como por el teléfono o subido de imágenes.

Cifrado Como recomendación para los JWT, al observar los resultados de la entropía sería mejor poder utilizar un JWT en donde la información también esté cifrada y no solo firmada. Ya que aunque puedan obtener el token, es más difícil poder obtener la información que se encuentra en el *payload*.

CAPÍTULO 11

Bibliografía

[1] *Types of APIs | Types Of API Calls & REST API Protocol*, 2022. https://stoplight.io/api-types.

[2] Academy, Binance: *¿Qué son las Criptomonedas?* https://academy.binance.com/es/start-here?utm_campaign=googleadsxacademy&utm_source=googleadwords_int&utm_medium=cpc&ref=HDYAHEES&gclid=Cj0KCQjwyYKUBhDJARIsAMj9lkFJMcQG0ebSuSJ7b3E0dpr2xfKuAlHkaGSTUVQzizYdsJI8\E8cSulkaAkyxEALw_wcB.

[3] Academy, Bit2Me: *¿Qué es el algoritmo de firma ECDSA?*, 2022. https://academy.bit2me.com/que-es-ecdsa-curva-eliptica/.

[4] amBientech: *Definición: Neurona*, Junio 2022. https://ambientech.org/la-neurona.

[5] Atlassian: *Bitbucket | Git solution for teams using Jira.* https://bitbucket.org.

[6] Atlassian: *Jira | Software de seguimiento de proyectos e incidencias.* https://www.atlassian.com/es/software/jira.

[7] Atom: *A hackable text editor for the 21st Century.* https://atom.io.

[8] Awati, R: *Hypertext Transfer Protocol Secure (HTTPS)*, Marzo 2022. https://www.techtarget.com/searchsoftwarequality/definition/HTTPS.

[9] AWS: *¿Qué es una red neuronal? Guía de IA y ML - AWS.* https://aws.amazon.com/es/what-is/neural-network/.

[10] AWS SDK S3: *aws-sdk-s3 | RubyGems.org | your community gem host.* https://rubygems.org/gems/aws-sdk-s3/versions/1.114.0.

[11] Banco Mundial: *Por qué es importante garantizar los derechos sobre la tierra*, Junio 2017. https://www.bancomundial.org/es/news/feature/2017/03/24/why-secure-land-rights-matter.

[12] Barrios, Juan: *Redes neuronales convolucionales.* Juan Barrios, 2022. https://www.juanbarrios.com/redes-neurales-convolucionales/.

[13] Brakeman: *brakeman | RubyGems.org | el alojamiento de gemas de tu comunidad.* https://rubygems.org/gems/brakeman/versions/3.4.0?locale=es.

[14] Brunner, Alexander E.: *SWISS DIGITAL ASSET AND WEALTH MANAGEMENT REPORT 2021*. CV VC AG, 2021.

[15] Bundler Audit: *GitHub - rubysec/bundler-audit: Patch-level verification for Bundler*. https://github.com/rubysec/bundler-audit.

[16] Burrueco, D.: *El perceptrón multicapa*. Interactivechaos, 2022. https://interactivechaos.com/es/manual/tutorial-de-deep-learning/el-perceptron-multicapa.

[17] Chohan, Usman W. y Usman W. Chohan: *Non-Fungible Tokens: Blockchains, Scarcity, and Value (en inglés)*. SSRN, 2021.

[18] Cloud, Google: *¿Qué es cloud computing?* https://cloud.google.com/learn/what-is-cloud-computing?hl=es.

[19] Code, VS: *Visual Studio Code - Code Editing. Redefined*, Noviembre 2021. https://code.visualstudio.com.

[20] Construir, Revista: *CEOs de la industria inmobiliaria no están interesados en tecnología*, 2018. https://revistaconstruir.com/ceos-industria-inmobiliaria-perdiendo-tecnologica/.

[21] D. Rodeck, B. Curry: *What Is Blockchain?*, Abril 2022. https://www.forbes.com/advisor/investing/cryptocurrency/what-is-blockchain/.

[22] DBeaver: *DBeaver Community | Free Universal Database Tool*. https://dbeaver.io.

[23] Devise: *GitHub - heartcombo/devise: Flexible authentication solution for Rails with Warden.* https://github.com/heartcombo/devise.

[24] Django: *The web framework for perfectionists with deadlines | Django*. https://www.djangoproject.com.

[25] Editor: *¿Cómo la tecnología ayuda a combatir el fraude en licencias y facturas médicas?* Mundo en Linea, 2022. https://mundoenlinea.cl/2022/05/11/como-la-tecnologia-ayuda-a-combatir-el-fraude-en-licencias-y-facturas-medicas/.

[26] Ernst & Young: *Non-fungible tokens (NFTs) in the spotlight*. Ernst & Young, Julio 2021.

[27] Etecé, Editorial: *Base de Datos - Concepto, tipos y ejemplos*, 2021. https://concepto.de/base-de-datos/.

[28] Ethereum: *Contract ABI Specification* . https://docs.soliditylang.org/en/v0.8.16/abi-spec.html.

[29] Ethereum: *Non-fungible tokens (NFT)*. https://ethereum.org/en/nft/.

[30] Ethereum: *What is Web3 and why is it important?* https://ethereum.org/en/web3/.

[31] ExpressJS: *Express - Infraestructura de aplicaciones web Node.js*. https://expressjs.com/es/.

[32] Feldman, J. & Rojas, R.: *Neural Networks: A Systematic Introduction (1 ed.)*. Springer, 1996.

[33] Fernández, Y: *Que son los NFT y cómo funcionan*, Julio 2022. https://www.xataka.com/basics/que-nft-como-funcionan.

[34] Finzer, Devin: *The Non-Fungible Token Bible: Everything you need to know about NFTs*, 2020. https://opensea.io/blog/guides/non-fungible-tokens/.

[35] Garg, M.: *11 Python frameworks for web development (en inglés)*. Net solutions, 2022. https://www.netsolutions.com/insights/top-10-python-frameworks-for-web-development-in-2019/#1-django.

[36] GeeksforGeeks: *XGBoost for Regression*, Noviembre 2022. https://www.geeksforgeeks.org/xgboost-for-regression/.

[37] Github: *GitHub: Let's build from here*. https://github.com.

[38] GitLab: *The One DevOps Platform*. https://about.gitlab.com.

[39] Harris, S. Maymi F: *CISSP All in one*. 2018.

[40] Harvard Business Review, family=Review given i=H.B., family=Tucker given i=C., family=Tapscott given i=D., family=Iansiti given i=M. y family=Lakhani given i=K.R.: *Blockchain*. Reed Business Education, Amsterdam, Netherlands, 2019.

[41] Haykin, S: *Neural Networks: A Comprehensive Foundation (2 Sub ed)*. Prentice Hall, 1998.

[42] Hays, Demelza, Solomon Guy, Alexander Valentin, Vladimir Shapovalov, Igor Kravchenko, Nikita Malkin, Ron Mendoza y Helen Rosenberg: *Nonfungible Tokens: A New Frontier*. Cointelegraph Research, 2021.

[43] Hussey, M.D.P.: *What is MetaMask? How to Use the Top Ethereum Wallet*, Junio 2022. https://decrypt.co/resources/metamask.

[44] Iberdrola: *Smart contracts: contratos inteligentes para formalizar acuerdos en la era digital*, 2022. https://www.iberdrola.com/innovacion/smart-contracts.

[45] IBM: *Política y objetivos de seguridad*. https://www.ibm.com/docs/es/i/7.3?topic=security-policy-objectives.

[46] IBM: *What are smart contracts on blockchain?* https://www.ibm.com/topics/smart-contracts.

[47] IBM: *What is a data set?* https://www.ibm.com/docs/en/zos-basic-skills?topic=more-what-is-data-set.

[48] IBM: *What is Blockchain Technology?* https://www.ibm.com/topics/what-is-blockchain.

[49] IBM: *Bagging (en inglés)*. IBM, 2022. https://www.ibm.com/cloud/learn/bagging.

[50] IBM: *Bagging (en inglés)*. IBM, 2022. https://www.ibm.com/cloud/learn/bagging.

[51] IBM: *Linear regression (en inglés)*. IBM, 2022. https://www.ibm.com/topics/linear-regression.

[52] IBM: *Random Forest*. IBM, 2022. https://www.ibm.com/cloud/learn/random-forest.

[53] IBM: *¿Qué es un árbol de decisión?* IBM, 2022. https://www.ibm.com/es-es/topics/decision-trees.

[54] IDnow: *What is KYC? Overview & short explanations*, Octubre 2022. https://www.idnow.io/regulation/what-is-kyc/.

[55] inglés), US Census Bureau (en: *AHS 2019 National Public Use File (PUF)*, Octubre 2021. https://www.census.gov/programs-surveys/ahs/data/2019/ahs-2019-public-use-file--puf-/ahs-2019-national-public-use-file--puf-.html.

[56] i=Y, family=Wolff given: *Optimize Response Time of your Machine Learning API in Production*. https://www.sicara.fr/blog-technique/optimize-response-time-api.

[57] JetBrains: *JetBrains: herramientas de desarrollador para profesionales y equipos.* https://www.jetbrains.com/es-es/.

[58] Jones, et al: *RFC 7519*. RFC, 2015. https://www.rfc-editor.org/rfc/rfc7519.

[59] K. Wegrzyn, E.Wang: *Types of Blockchain: Public, Private, or Something in Between*, Agosto 2021. https://www.foley.com/en/insights/publications/2021/08/types-of-blockchain-public-private-between.

[60] KeepCoding: *¿Qué es RSA en criptografía?*, 2022. https://keepcoding.io/blog/que-es-rsa-en-criptografia/.

[61] Krizhevsky, A, I. Sutskever y G.E. Hinton: *ImageNet Classification with Deep Convolutional Neural Networks (en inglés).* https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf, visitado el 2022-11-02.

[62] Laravel: *Laravel - The PHP Framework For Web Artisans.* https://laravel.com.

[63] Levi, S. y A. Lipton: *An Introduction to Smart Contracts and Their Potential and Inherent Limitations*, Mayo 2018. https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/.

[64] Library, The Python Standard: *random — Generate pseudo-random numbers — Python 3.11.0 documentation.* https://docs.python.org/3/library/random.html.

[65] Lutkevich, B.: *framework*, Agosto 2020. https://www.techtarget.com/whatis/definition/framework.

[66] Microsoft: *HMACSHA256 Clase (System.Security.Cryptography).* https://learn.microsoft.com/es-es/dotnet/api/system.security.cryptography.hmacsha256?view=net-6.0.

[67] Microsoft Teams: *Videoconferencia, reuniones, llamadas | Microsoft Teams.* https://www.microsoft.com/es/microsoft-teams/group-chat-software/.

[68] Minsky, M y S Papert: *Perceptrons.* En *Perceptrons*, 1969, ISBN 0 262 13043 2.

[69] Monday: *monday.com: una alternativa para la gestión de proyectos.* https://monday.com/lp/lang/es/comp/alternative/remote/?utm_medium=cpc.

[70] Mozilla: *Data URLs*, Septiembre 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URLs.

[71] Mozilla: *FormData - Web APIs | MDN*, Octubre 2022. https://developer.mozilla.org/en-US/docs/Web/API/FormData.

[72] Mozilla: *Frameworks Web de lado servidor - Aprende sobre desarrollo web | MDN*, Octubre 2022. https://developer.mozilla.org/es/docs/Learn/Server-side/First_steps/Web_frameworks.

[73] Mozilla: *POST - HTTP | MDN*, Septiembre 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST.

[74] Mozilla: *Que es un servidor WEB? - Aprende sobre desarrollo web | MDN*, Septiembre 2022. https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_web_server.

[75] Mozilla: *Trabajando con JSON* , Octubre 2022. https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON.

[76] Mozilla: *Visión General Cliente-Servidor - Aprende sobre desarrollo web | MDN*, Septiembre 2022. https://developer.mozilla.org/es/docs/Learn/Server-side/First_steps/Client-Server_overview.

[77] MySQL: *MySQL*. https://www.mysql.com.

[78] OpenSea: *20Mission*. https://opensea.io/es/collection/20mission.

[79] Oracle: *¿Qué es una base de datos?* https://www.oracle.com/mx/database/what-is-database/.

[80] OWASP: *SQL Injection | OWASP Foundation*. https://owasp.org/www-community/attacks/SQL_Injection.

[81] P., Nwadike: *Industria inmobiliaria: descripción general, tipos y ejemplos*, Agosto 2021. https://businessyield.com/es/real-estate-investment/real-estate-industry/.

[82] PG: *pg | RubyGems.org | el alojamiento de gemas de tu comunidad*. https://rubygems.org/gems/pg/versions/0.18.4?locale=es.

[83] Portswigger: *Burp Suite - Application Security Testing Software*. https://portswigger.net/burp.

[84] PostgreSQL: *PostgreSQL*. https://www.postgresql.org.

[85] Postman: *Postman*. https://www.postman.com.

[86] Propy Inc.: *Propy | Real Estate Transaction Automated*, Noviembre 2022. https://propy.com/browse/.

[87] R., Poole D. & Mackworth A. & Goebel: *Computational Intelligence: A Logical Approach (en inglés)*. Oxford University Press, 1998, ISBN 978-0-19-510270-3.

[88] Rack Cors: *rack-cors | RubyGems.org | el alojamiento de gemas de tu comunidad*. https://rubygems.org/gems/rack-cors/versions/0.4.0?locale=es.

[89] React: *Un vistazo a los Hooks*. https://es.reactjs.org/docs/hooks-overview.html.

[90] RedHat: *What is a REST API?* https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[91] RedHat: *¿Qué es una API?*, 2017. https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces.

[92] Rubocop: *GitHub - rubocop/rubocop: A Ruby static code analyzer and formatter, based on the community Ruby style guide*. https://github.com/rubocop/rubocop.

[93] Ruby JWT: *GitHub - jwt/ruby-jwt: A ruby implementation of the RFC 7519 OAuth JSON Web Token (JWT) standard*. https://github.com/jwt/ruby-jwt.

[94] Ruby on Rails: *Ruby on Rails - Compress the complexity of modern web apps*. https://rubyonrails.org.

[95] Ruizendaal, R.: *Deep Learning #3: More on CNNs & Handling Overfitting*, Mayo 2017. https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d.

[96] Russel, Stuart J.; Norvig, Peter Norvig: *Artificial intelligence: a modern approach (en inglés)*. En *Artificial intelligence: a modern approach (en inglés)*, 2009, ISBN 0-13-604259-7.

[97] RVM: *RVM: Ruby Version Manager - RVM Ruby Version Manager - Documentation*. https://rvm.io.

[98] Sanchez, J.A.: *¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos*, Diciembre 2020. https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos.

[99] Sharma, G.: *5 Regression Algorithms you should know.* Analytics Vidhya, 2022. https://www.analyticsvidhya.com/blog/2021/05/5-regression-algorithms-you-should-know-introductory-guide/.

[100] Singh, H.: *REST API security threats.* The Cyphere, 2022. https://thecyphere.com/blog/top-api-security-risks-and-attack-prevention/.

[101] Slack: *Slack es tu sede digital.* https://slack.com/intl/es-gt.

[102] Sociales, R.: *Conceptos básicos de Ciberseguridad: Triada CIA*, Febrero 2021. https://www.uniat.edu.mx/conceptos-basicos-de-ciberseguridad/.

[103] Stefanoski, Darko, Orkan Sahin, Benjamin Banusch, Stephanie Fuchs, Silvan Andermatt y Alexandre Quertramp: *Tokenization of Assets*, volumen 1. Ernst & Young Ltd, 2020.

[104] Stoplight: *Types of APIs.* Stop Light, 2022. https://stoplight.io/api-types.

[105] Takyar, Akash: *Token standards: ERC20 vs ERC721 vs ERC1155*, Septiembre 2022. https://www.leewayhertz.com/erc-20-vs-erc-721-vs-erc-1155/.

[106] Telegram: *Telegram – a new era of messaging.* https://telegram.org.

[107] Text, Sublime: *Sublime Text - the sophisticated text editor for code, markup and prose.* https://www.sublimetext.com.

[108] The FreeBSD Project: *Explaining BSD*, 1994. https://docs.freebsd.org/en/articles/explaining-bsd/.

[109] Trello: *Manage Your Teams Projects From Anywhere | Trello.* https://trello.com.

[110] WSGI: *What is WSGI? — WSGI.org.* https://wsgi.readthedocs.io/en/latest/what.html.

CAPÍTULO 12

ANEXO A

Administración del proyecto

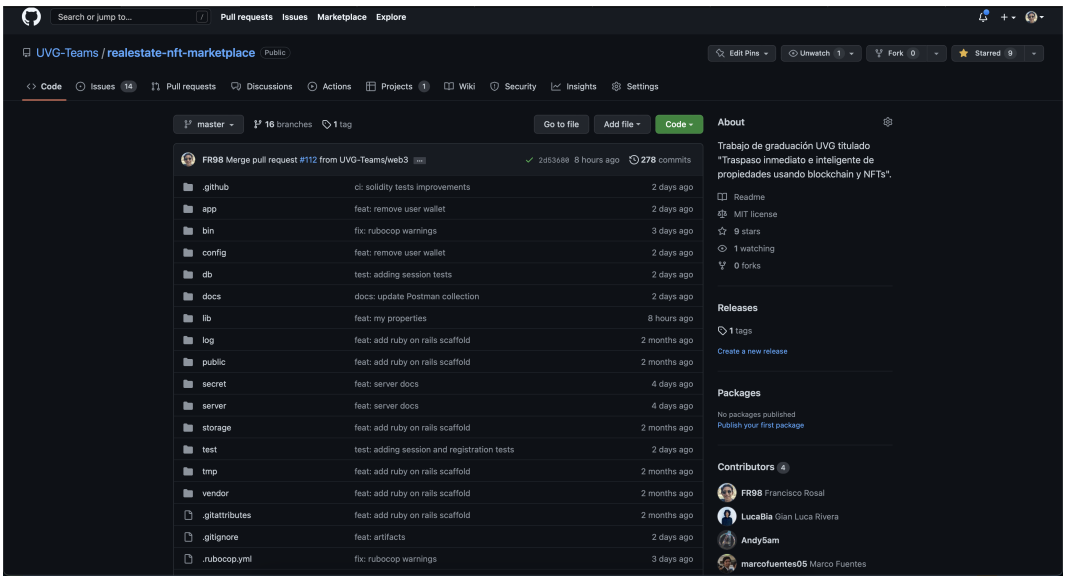


Figura A.1: Repositorio en Github.

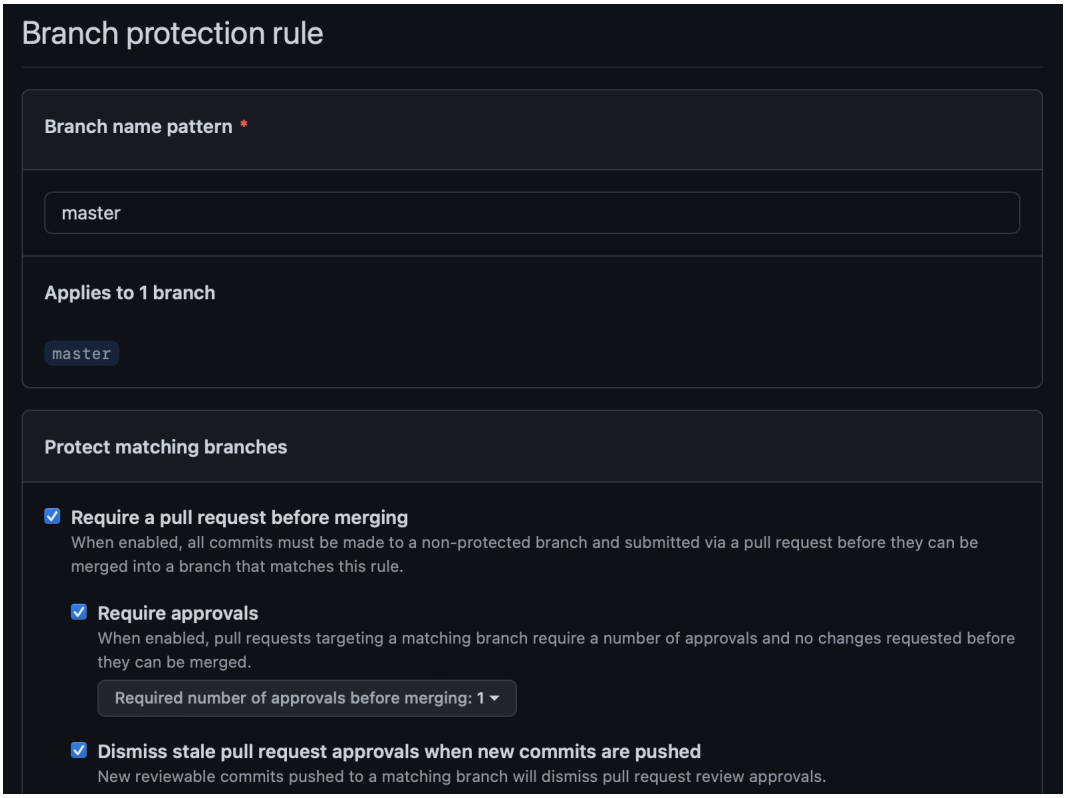


Figura A.2: Configuración de protección de ramas 1.

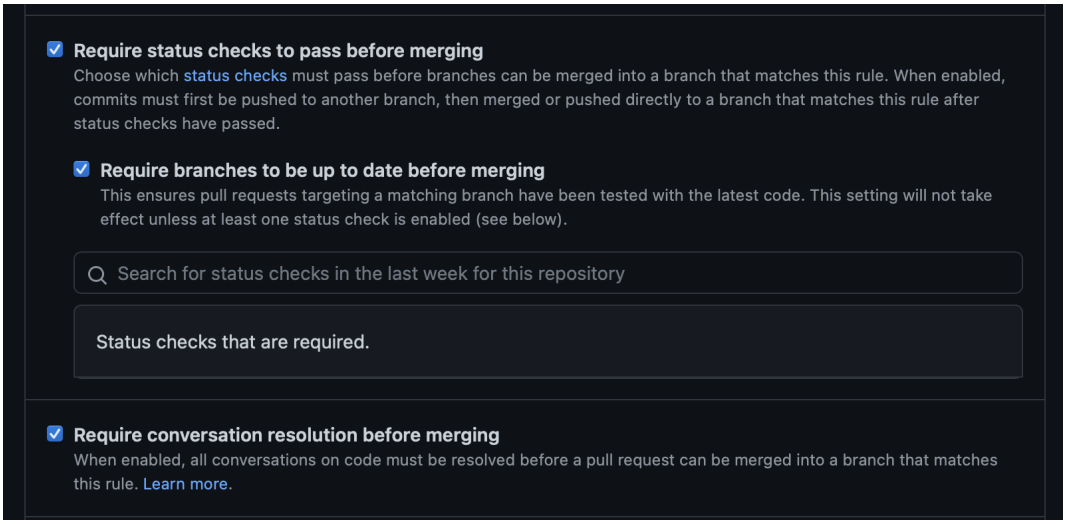


Figura A.3: Configuración de protección de ramas 2.

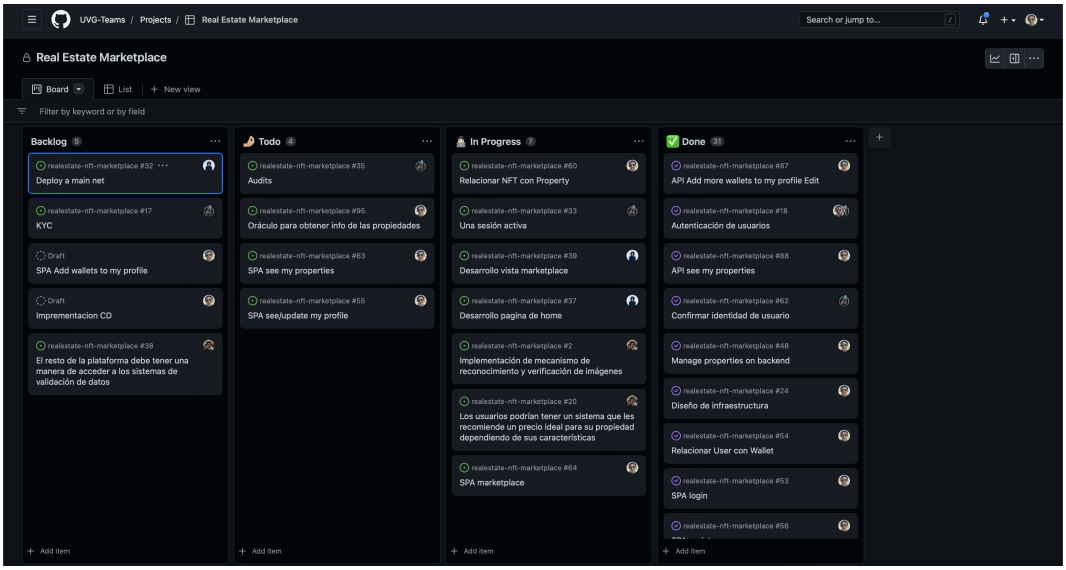


Figura A.4: Github Project.

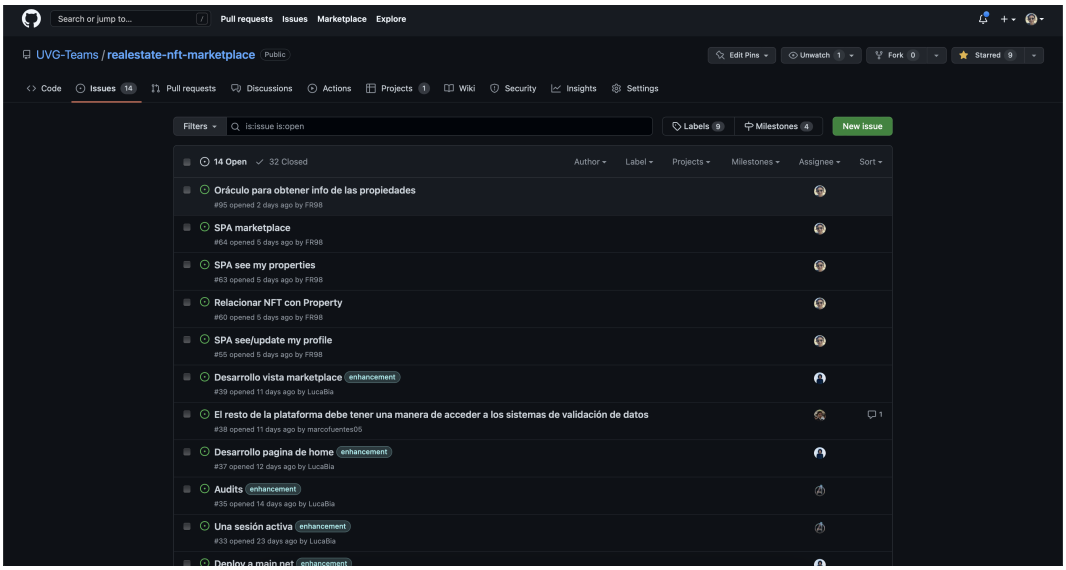


Figura A.5: Github Issues.

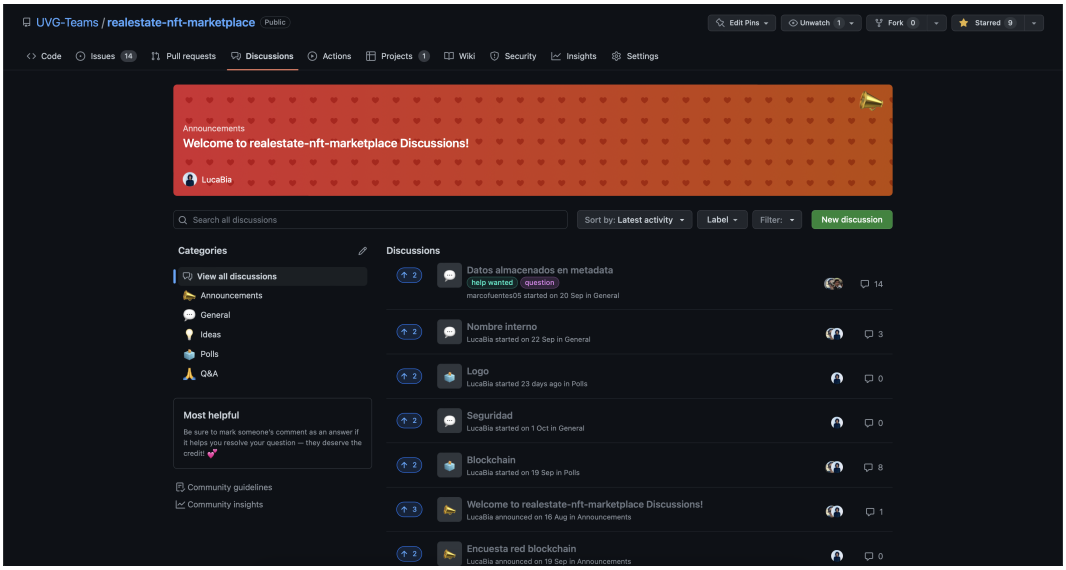


Figura A.6: Github Discussions.

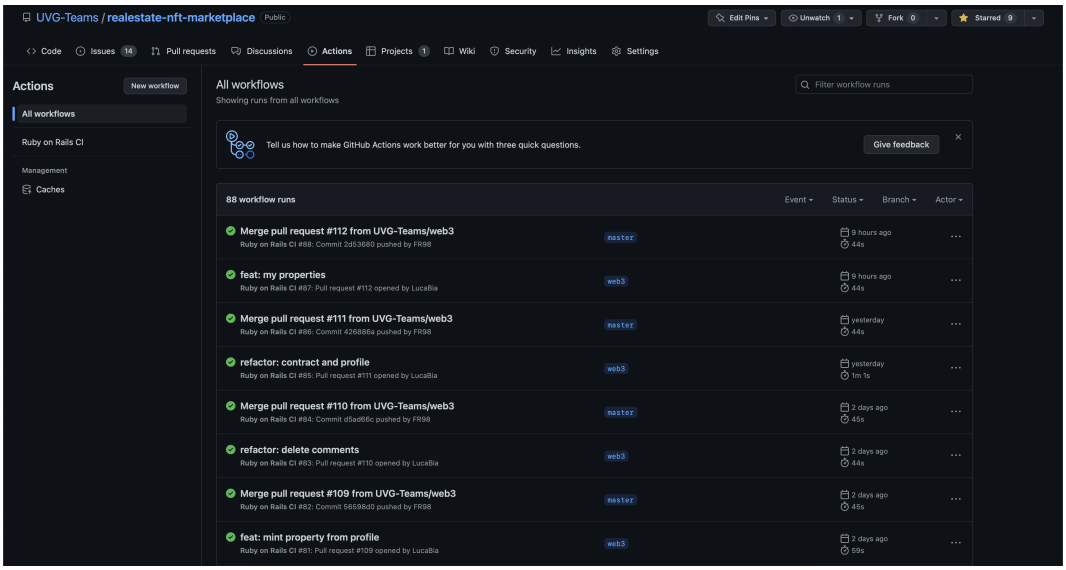


Figura A.7: Github Actions.

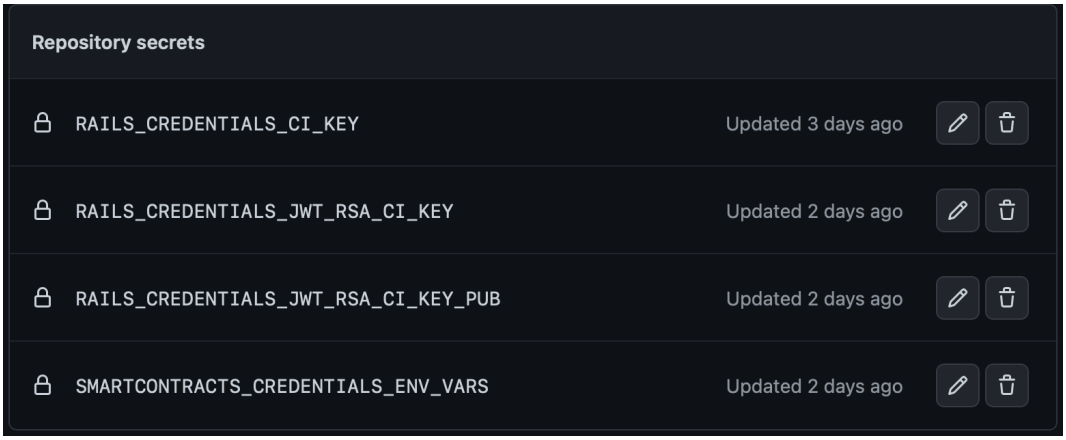


Figura A.8: Github Action Secrets.

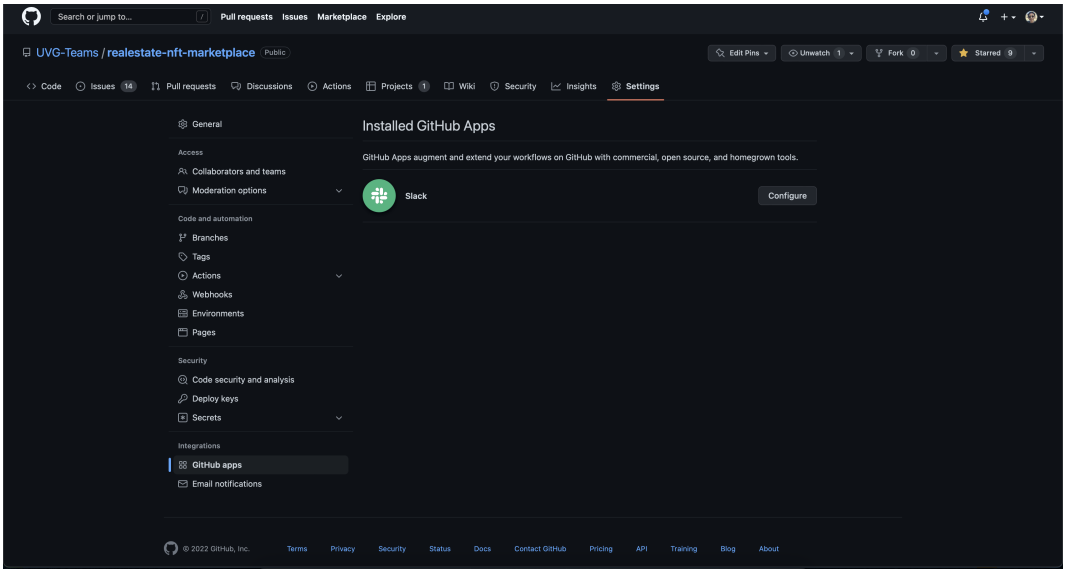


Figura A.9: Github Apps.

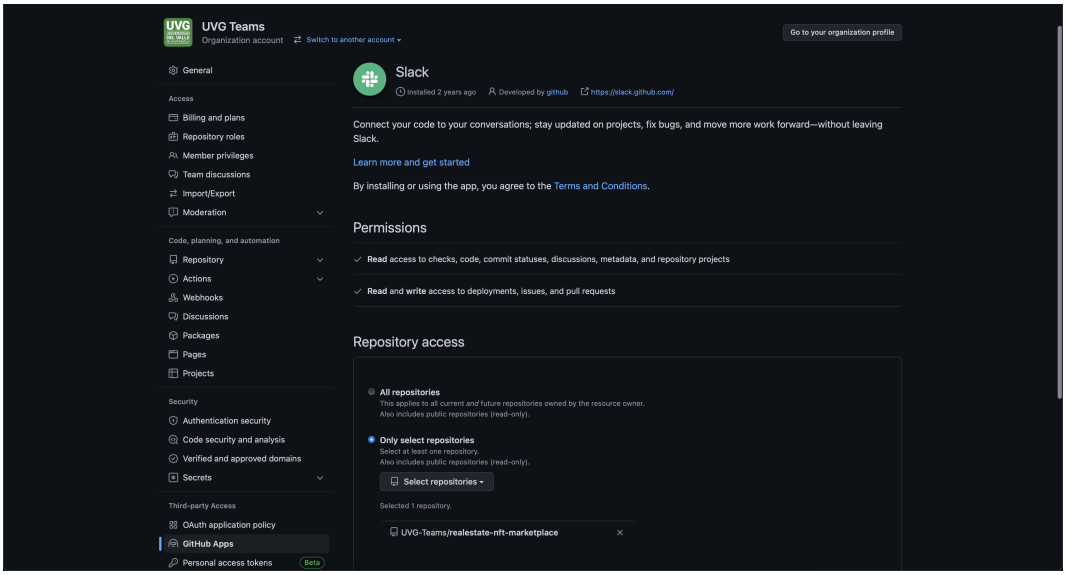


Figura A.10: Github Apps - Integración con Slack.

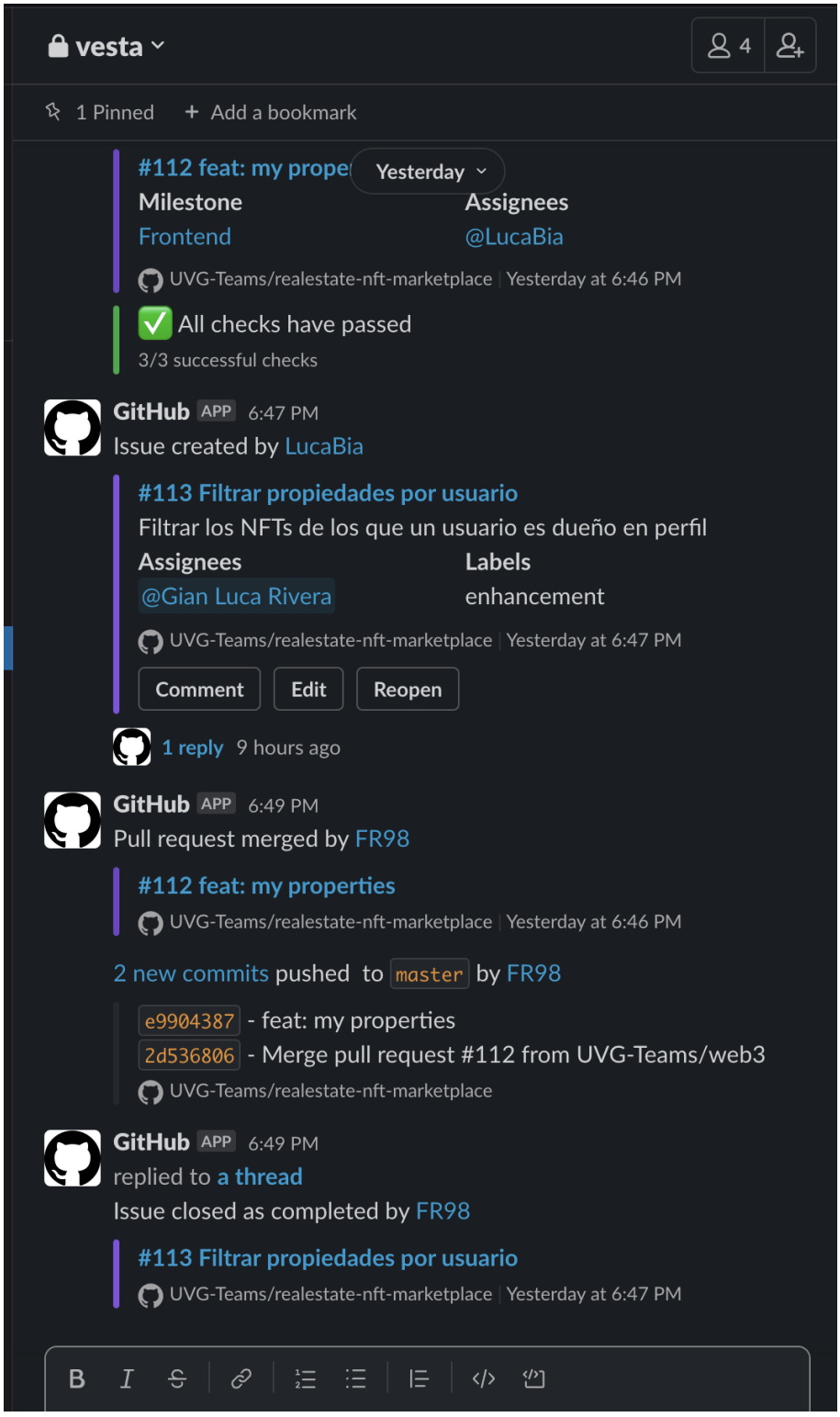


Figura A.11: Slack.

ANEXO B

Investigaciones

B.1. HMAC

El código de autenticación de mensajes basado en hash o HMAC es una técnica de autenticación criptográfica que utiliza una función hash y una clave secreta. Con HMAC se puede lograr la autenticación y verificar que los datos sean correctos y auténticos con secretos compartidos, a diferencia de los enfoques que usan firmas y criptografía asimétrica. HMAC consiste de dos partes, las llaves criptográficas, ya que un algoritmo de cifrado altera los datos y el destinatario necesita un código o llave específico para volver a leerlos. HMAC se basa en conjuntos compartidos de claves secretas. La otra parte es una función hash, un algoritmo hash altera o digiere el mensaje una vez más. Las entidades deben acordar tanto la llave como el algoritmos a usar[66].

B.2. RSA

El algoritmo RSA es un algoritmo de criptografía asimétrica, es decir necesita de una llave pública y una privada. La idea de RSA se basa en el hecho de que es difícil factorizar un número entero grande. La clave pública consiste de dos números donde un número es una multiplicación de dos números primos grandes. Y la clave privada también se deriva de los mismos dos números primos. Entonces, si alguien puede factorizar el gran número, la clave privada está comprometida. Por lo tanto, la fuerza del cifrado depende totalmente del tamaño de la clave [60].

B.3. ECDSA

El algoritmo de firma digital de curva elíptica (ECDSA) es un algoritmo de firma digital que utiliza claves derivadas de la criptografía de curva elíptica, así como el algoritmo RSA usa una llave pública y una privada, sin embargo estas son calculadas con fórmulas diferentes. Es una ecuación particularmente eficiente basada en criptografía de clave pública. ECDSA se usa en muchos sistemas

de seguridad, es popular para su uso en aplicaciones de mensajería segura y es la base de la seguridad de Bitcoin [3].

B.4. Burp Suite

Burp Suite es una plataforma la cual nos permite realizar varias pruebas de penetración a través de las herramientas que tiene. En este caso la herramienta que se utilizó fue el secuenciador que es un verificador de entropía que verifica la aleatoriedad de los tokens generados por el servidor web. Estos tokens se utilizan generalmente para la autenticación en operaciones confidenciales. Idealmente, estos tokens deben generarse de manera totalmente aleatoria para que la probabilidad de aparición de cada posible carácter en una posición se distribuya uniformemente [83].

ANEXO C

Scripts

C.1. API *Controllers*

```
class Api::Users::RegistrationController < ApplicationApiController
    skip_before_action :authorize_request, only: :create

    def create
        @current_user = nil

        return respond_with_status(400, 'Email required') if registration_params[:email].blank?
        return respond_with_status(400, 'Password required') if registration_params[:password].blank?
        return respond_with_status(400, 'Password confirmation required') if registration_params[:password_confirmation].blank?

        # Looking for the user
        user = User.find_for_database_authentication(email: registration_params[:email])

        # User already exists
        return respond_with_status(400, 'User already exists') unless user.blank?

        # Password complexity mismatched
        return respond_with_status(400, 'Password must contain lower case, upper case and number') if registration_params[:params].blank? && !registration_params[:password].match(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)./)

        # Pasword mismatch
        return respond_with_status(400, 'Pasword mismatch') if registration_params[:password] != registration_params[:password_confirmation]

        # Setting @current_user
        @current_user = User.new(registration_params)
        @current_user.active = true

        if @current_user.save

            if wallet_params[:account]
```

```ruby

                # Look account in all registered
                wallet = User::Wallet.find_by(account: wallet_params[:account])

                # Account already registered
                return respond_with_status(400, 'Account already registered by other user') if wallet

                # Registering account for the current_user
                @current_user.wallets.create({
                    account: wallet_params[:account]
                })

            end

            # Generate JWT
            auth_token = ::AuthToken.new(@current_user)

            respond_with_status(200, {
                token: auth_token.token
            })

        else
            respond_with_status(400, 'Try again later')
        end
    end

    private

    def registration_params
        params.fetch(:registration, {}).permit(
            :email,
            :password,
            :password_confirmation,
            :first_name,
            :last_name,
            :telephone
        )
    end

    def wallet_params
        params.fetch(:wallet, {}).permit(:account)
    end
end
```

Script C.1: (Ruby) app/controllers/api/users/registration_controller.rb

```ruby
class Api::Users::SessionsController < ApplicationApiController
    skip_before_action :authorize_request, only: :create

    def create
        return respond_with_status(400, 'Password required') if session_params[:password].blank?

        if session_params[:account]

            wallet = User::Wallet.find_by(account: session_params[:account])

            return respond_with_status(404, 'Wallet account not found') unless wallet

            # Setting @current_user
            @current_user = wallet.user

        else

            # Looking for the user
            user = User.find_for_database_authentication(email: session_params[:email])

            # User not found
            return respond_with_status(404, 'User not found') if user.blank?

            # Setting @current_user
            @current_user = user

        end

        # Validate password
        return respond_with_status(400, 'Wrong password') unless @current_user.valid_password?(session_params[:password])

        # Generate JWT
        auth_token = ::AuthToken.new(@current_user, {
            email: @current_user.email,
            first_name: @current_user.first_name,
            last_name: @current_user.last_name,
            telephone: @current_user.telephone
        })

        respond_with_status(200, {
            token: auth_token.token
        })
    end

    private

    def session_params
        params.fetch(:session, {}).permit(:email, :password, :account)
    end
end
```

Script C.2: (Ruby) app/controllers/api/users/sessions_controller.rb

```ruby
class Api::PropertiesController < ApplicationApiController
    before_action :set_property, only: %i[show edit update destroy get_files
    upload_files]
    before_action :check_ownership, only: %i[update destroy get_files upload_files]

    # GET /properties or /properties.json
    def index
        @properties = Property.all
        respond_with_status(200, @properties)
    end

    # GET /properties/1 or /properties/1.json
    def show
        respond_with_status(200, @property)
    end

    # POST /properties or /properties.json
    def create
        begin
            @property = Property.new(property_params)
        rescue StandardError => e
            return respond_with_status(400, e.to_s)
        end

        @property.user = @current_user

        if @property.save
            respond_with_status(200, 'Property was successfully created.')
        else
            respond_with_status(400, @property.errors)
        end
    end

    # PATCH/PUT /properties/1 or /properties/1.json
    def update
        if @property.update(property_params)
            respond_with_status(200, 'Property was successfully updated.')
        else
            respond_with_status(400, @property.errors)
        end
    end

    # DELETE /properties/1 or /properties/1.json
    def destroy
        @property.destroy

        respond_with_status(200, 'Property was successfully destroyed.')
    end

    # ============================================================
    # Custom methods
    # ============================================================

    def retrieve_files
        response = {
            images: []
        }

        @property.files.each do |file|
            response[:images].append({
                id: file.id,
                url: url_for(file)
            })
        end

        respond_with_status(200, response)
    end

```

```ruby
    def upload_files
        # Attach the new file
        @property.files.attach(params[:files])

        response = {
            images: []
        }

        @property.files.each do |file|
            response[:images].append({
                id: file.id,
                url: url_for(file)
            })
        end

        respond_with_status(200, response)
    end

    private

    # Use callbacks to share common setup or constraints between actions.
    def set_property
        @property = Property.find_by_id(params[:id])
        return respond_with_status(404, 'Property not found.') if @property.blank?
    end

    # Only allow a list of trusted parameters through.
    def property_params
        params.fetch(:property, {}).permit(
            :finca,
            :folio,
            :libro,
            :location,
            :category,
            :rooms,
            :bathrooms
        )
    end

    def check_ownership
        return respond_with_status(401) if @current_user != @property.user
    end
end
```

Script C.3: (Ruby) app/controllers/api/properties_controller.rb

```ruby
class Api::UsersController < ApplicationApiController
    before_action :set_user, only: %i[show update destroy properties upload_avatar remove_wallet]
    before_action :check_ownership, only: %i[update destroy upload_avatar remove_wallet]

    # GET /users or /users.json
    def index
        @users = User.all
        respond_with_status(200, @users)
    end

    # GET /users/1 or /users/1.json
    def show
        respond_with_status(200, @user.slice(
            :id,
            :email,
            :active,
            :created_at,
            :updated_at,
            :first_name,
            :last_name,
            :telephone,
            :pid_number
        ).merge(
            avatar: @user.avatar.attached? ? url_for(@user.avatar) : nil,
            wallets: @user.wallets
        ))
    end

    # PATCH/PUT /users/1 or /users/1.json
    def update
        if @user.update(user_params)
            respond_with_status(200, 'User was successfully updated.')
        else
            respond_with_status(400, @user.errors)
        end
    end

    # DELETE /users/1 or /users/1.json
    def destroy
        @user.destroy

        respond_with_status(200, 'User was successfully destroyed.')
    end

    # ============================================================
    # Custom methods
    # ============================================================

    def upload_avatar
        # Remove the previous if exists
        @current_user.avatar.purge if @current_user.avatar.attached?

        # Attach the new file
        @current_user.avatar.attach(params[:avatar])

        respond_with_status(200, {
            avatar_url: url_for(@current_user.avatar)
        })
    end

    def properties
        user_properties = @user.properties.all
        respond_with_status(200, user_properties)
    end

    def wallet
```

```ruby
        return respond_with_status(400) if wallet_params[:account].blank?

        # Look account in all registered
        wallet = User::Wallet.find_by(account: wallet_params[:account])

        # Account already registered
        return respond_with_status(400, 'Account already registered by other user') if wallet

        wallet = @current_user.wallets.new(wallet_params)

        if wallet.save
            respond_with_status(200, 'Wallet was successfully created.')
        else
            respond_with_status(400, wallet.errors)
        end
    end

    def remove_wallet
        return respond_with_status(400) if wallet_params[:account].blank?

        wallet = @user.wallets.find_by(account: wallet_params[:account])

        return respond_with_status(404) unless wallet

        if wallet.destroy
            respond_with_status(200, 'Wallet was successfully destroy.')
        else
            respond_with_status(400, wallet.errors)
        end
    end

    private

    # Use callbacks to share common setup or constraints between actions.
    def set_user
        @user = User.find_by_id(params[:id])
        return respond_with_status(404, 'User not found.') if @user.blank?
    end

    # Only allow a list of trusted parameters through.
    def user_params
        params.fetch(:user, {})
    end

    def wallet_params
        params.fetch(:wallet, {}).permit(:account)
    end

    def check_ownership
        return respond_with_status(401) if @current_user != @user
    end
end
```

Script C.4: (Ruby) app/controllers/api/users_controller.rb

C.2. API *Helper*

```ruby
module ApplicationApiHelper
    ANSI_RED = "\e[31m".freeze
    ANSI_WHITE = "\e[37m".freeze
    ANSI_RESET = "\e[0m".freeze

    def respond_with_status(status = 200, payload = {})
        payload = { msg: payload } if payload.is_a?(String)

        case status
        when 200
            payload[:msg] = '200 - Ok' unless payload
        when 400
            payload[:msg] = '400 - Bad Request' unless payload[:msg]
        when 401
            payload[:msg] = '401 - Not Authorized' unless payload[:msg]
        when 404
            payload[:msg] = '404 - Not Found' unless payload[:msg]
        end

        render(status: status, content_type: 'application/json', json: payload.
    to_json)
    end

    def debug(*args)
        puts ANSI_RED
        puts '-' * 100
        args.each do |arg|
            puts arg
        end
        puts '-' * 100
        puts ANSI_RESET
    end
end
```

Script C.5: (Ruby) app/helpers/application_api_helper.rb

C.3. *Routes*

```ruby
Rails.application.routes.draw do
    devise_for :users,
    path: '',
    path_names: {
        sign_in: 'login',
        sign_out: 'logout',
        sign_up: 'register'
    }

    resources :users, only: [:index]

    namespace :api do
        namespace :users do
            resources :sessions, only: [:create]
            resources :registration, only: [:create]
        end

        resources :users do
            scope module: :user do
                resources :wallets, only: []
            end

            member do
                get     :properties
                post    :wallet
                delete  :remove_wallet
                post    :upload_avatar
            end
        end

        resources :properties do
            member do
                post    :upload_files
                get     :retrieve_files
            end
        end
    end
end
```

Script C.6: (Ruby) config/routes.rb

C.4. *Services*

```ruby
require 'jwt'
# Docs: https://github.com/jwt/ruby-jwt

class AuthToken
    attr_accessor :token

    @@algorithm = 'RS256'

    def initialize(current_user, custom_payload = {})
        @current_user = current_user
        @custom_payload = custom_payload

        generate
    end

    def generate
        private_key_path = Rails.root.join('config/credentials/jwtRS256.key')
        private_key_file = File.open(private_key_path)

        private_key_data = private_key_file.read

        private_key = OpenSSL::PKey::RSA.new(private_key_data)

        # Setting the payload of the token
        payload = {
            **@custom_payload,
            sub: @current_user.id,
            # exp: 24.hours.from_now.to_i,
            exp: 5.minutes.from_now.to_i
        }

        self.token = JWT.encode payload, private_key, @@algorithm
    end

    def self.verify(token)
        public_key_path = Rails.root.join('config/credentials/jwtRS256.key.pub')

        public_key_file = File.open(public_key_path)

        public_key_data = public_key_file.read

        public_key = OpenSSL::PKey::RSA.new(public_key_data)

        begin
            decoded_token = JWT.decode token, public_key, true, { algorithm:
    @@algorithm }
        rescue StandardError => e
            Debugger.debug e
            return false, nil
        end

        # Decoding parts of the token
        payload = decoded_token[0]
        header = decoded_token[1]

        # Check if the algorithm is correct
        return false, nil if header['alg'] != @@algorithm

        # Check if token is expired
        return false, nil if 0.seconds.from_now.to_i > payload['exp'].to_i

        [true, payload]
    end

    def self.refresh(token)
```

```
65          valid, payload = self.verify(token)
66
67          return nil unless valid
68
69          current_user = User.find_by_id(payload['sub'])
70          custom_payload = payload.except('sub', 'exp')
71
72          self.new(current_user, custom_payload)
73      end
74
75      def self.generate_rsa
76          rsa_private = OpenSSL::PKey::RSA.generate 2048
77          rsa_public = rsa_private.public_key
78
79          Debugger.debug rsa_private
80          Debugger.debug rsa_public
81      end
82  end
```

Script C.7: (Ruby) app/services/auth_token.rb

C.5. *Testing*

```ruby
# frozen_string_literal: true

require 'test_helper'

class Api::Users::RegistrationControllerTest < ActionDispatch::IntegrationTest
    test 'should register with basic data' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email,
                password: 'Password1',
                password_confirmation: 'Password1'
            }
        }

        assert_response :success
    end

    test 'should register with basic data and account' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email,
                password: 'Password1',
                password_confirmation: 'Password1'
            },
            wallet: {
                account: Faker::Crypto.sha256
            }
        }

        assert_response :success
    end

    test 'should register with complete data' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email,
                password: 'Password1',
                password_confirmation: 'Password1',
                first_name: 'Admin',
                last_name: '3',
                telephone: '+50212345678'
            }
        }

        assert_response :success
    end

    test 'should register with complete data and account' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email,
                password: 'Password1',
                password_confirmation: 'Password1',
                first_name: 'Admin',
                last_name: '4',
                telephone: '+50212345678'
            },
            wallet: {
                account: Faker::Crypto.sha256
            }
        }

        assert_response :success
    end

```

```ruby
    test 'should not register with existent email' do
        post api_users_registration_index_path, params: {
            registration: {
                email: 'admin@vesta.com',
                password: 'Password1',
                password_confirmation: 'Password1'
            }
        }

        assert_response 400
    end

    test 'should not register without email' do
        post api_users_registration_index_path, params: {
            registration: {
                password: 'Password1',
                password_confirmation: 'Password1'
            }
        }

        assert_response 400
    end

    test 'should not register without password' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email
            }
        }

        assert_response 400
    end

    test 'should not register without matching passwords' do
        post api_users_registration_index_path, params: {
            registration: {
                email: Faker::Internet.email,
                password: 'Password1',
                password_confirmation: 'Password2'
            }
        }

        assert_response 400
    end
end
```

Script C.8: (Ruby) registration_controller_test.rb

```ruby
# frozen_string_literal: true

require 'test_helper'

class Api::Users::SessionsControllerTest < ActionDispatch::IntegrationTest
    test 'should login with email' do
        post api_users_sessions_path, params: {
            session: {
                email: 'admin@vesta.com',
                password: 'Password1'
            }
        }

        assert_response :success
    end

    test 'should login with account' do
        post api_users_sessions_path, params: {
            session: {
                account: '0x123admin@vesta.com',
                password: 'Password1'
            }
        }

        assert_response :success
    end

    test 'should not email login with wrong password' do
        post api_users_sessions_path, params: {
            session: {
                email: 'admin@vesta.com',
                password: 'Wrong_password1'
            }
        }

        assert_response 400
    end

    test 'should not account login with wrong password' do
        post api_users_sessions_path, params: {
            session: {
                account: '0x123admin@vesta.com',
                password: 'Wrong_password1'
            }
        }

        assert_response 400
    end

    test 'should login with wrong email' do
        post api_users_sessions_path, params: {
            session: {
                email: 'admin_not_found@vesta.com',
                password: 'Password1'
            }
        }

        assert_response 404
    end

    test 'should login with wrong account' do
        post api_users_sessions_path, params: {
            session: {
                account: '0x123adminadmin_not_found@vesta.com',
                password: 'Password1'
            }
        }

```

```ruby
        assert_response 404
    end

    test 'should not login without params' do
        post api_users_sessions_path, params: {}

        assert_response 400
    end

    test 'should not login with empty params' do
        post api_users_sessions_path, params: {
            session: {
                account: '',
                password: ''
            }
        }

        assert_response 400
    end

    test 'should not login with nil params' do
        post api_users_sessions_path, params: {
            session: {
                account: nil,
                password: nil
            }
        }

        assert_response 400
    end
end
```

Script C.9: (Ruby) sessions_controller_test.rb

C.6. CI *Workflow*

```yaml
# This workflow will install a prebuilt Ruby version, install dependencies, and
# run tests and linters.
name: "CI"
on:
  push:
    branches: [ "master", "production" ]
  pull_request:
    branches: [ "master", "production" ]
jobs:
  rails-test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:11-alpine
        ports:
          - "5432:5432"
        env:
          POSTGRES_DB: rails_test
          POSTGRES_USER: rails
          POSTGRES_PASSWORD: password
    env:
      RAILS_ENV: test
      RAILS_MASTER_KEY: ${{ secrets.RAILS_CREDENTIALS_CI_KEY }}
      DATABASE_URL: "postgres://rails:password@localhost:5432/rails_test"
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      # Add or replace dependency steps here
      - name: Install Ruby and gems
        uses: ruby/setup-ruby@0a29871fe2b0200a17a4497bae54fe5df0d973aa # v1.115.3
        with:
          bundler-cache: true
      # Add or replace database setup steps here
      - name: Bundle Install
        run: bundle install
      - name: Set up database
        run: |
          cp ./config/credentials/ci.yml.enc ./config/credentials/test.yml.enc
          echo "$PRIVATE_KEY" >> ./config/credentials/jwtRS256.key
          echo "$PUBLIC_KEY" >> ./config/credentials/jwtRS256.key.pub
          bundle exec rake db:drop
          bundle exec rake db:create
          bundle exec rake db:migrate
          bundle exec rake db:seed
        env:
          PRIVATE_KEY: ${{ secrets.RAILS_CREDENTIALS_JWT_RSA_CI_KEY }}
          PUBLIC_KEY: ${{ secrets.RAILS_CREDENTIALS_JWT_RSA_CI_KEY_PUB }}
      # Add or replace test runners here
      - name: Run tests
        run: bin/rake

  smartcontract-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      # Add or replace dependency steps here
      - name: Install Node
        uses: actions/setup-node@v3
        with:
          node-version: 16
      - name: NPM Install
        working-directory: ./lib/smartcontract
        run: npm install
      - name: Configuration of env
```

```
        working-directory: ./lib/smartcontract
        run: echo "$ENV_VARS" >> .env
        env:
          ENV_VARS: ${{ secrets.SMARTCONTRACTS_CREDENTIALS_ENV_VARS }}
      - name: Run tests
        working-directory: ./lib/smartcontract
        run: REPORT_GAS=true npx hardhat test

  lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Install Ruby and gems
        uses: ruby/setup-ruby@0a29871fe2b0200a17a4497bae54fe5df0d973aa # v1.115.3
        with:
          bundler-cache: true
      # Add or replace any other lints here
      - name: Security audit dependencies
        run: bundle exec bundler-audit
      - name: Security audit application code
        run: bundle exec brakeman
      - name: Lint Ruby files
        run: bundle exec rubocop --parallel
```

Script C.10: (.yml) CI Workflow

C.7. *Hook* obtención de *NFTs* almacenados en *blockchain*

```
import { useWeb3React } from "@web3-react/core";
import { useCallback, useEffect, useState } from "react";
import useVesta from "../useVesta";


const getVestaData = async ({ vesta, tokenId }) => {
    const [tokenURI, owner, houseDNA] = await Promise.all([
        vesta.methods.tokenURI(tokenId).call(),
        vesta.methods.ownerOf(tokenId).call(),
    ]);

    const respondeMetadata = await fetch(tokenURI);
    const metadata = await respondeMetadata.json();

    return {
        tokenId,
        // attributes: {
        //     houseDNA,
        // },
        tokenURI,
        owner,
        ...metadata
    }
}

// Obtencion conjunta
const useVestasData = ({ owner = null } = {}) => {
    const [allVestas, setVestas] = useState([]);
    const { library } = useWeb3React();
    const [loading, setLoading] = useState(true);
    const vesta = useVesta();

    const update = useCallback(async () => {
        if (vesta) {
            setLoading(true);

            let tokenIds;

            if (!library.utils.isAddress(owner)) {
                const totalSupply = await vesta.methods.totalSupply().call();
                tokenIds = new Array(Number(totalSupply)).fill().map((_, index) =>
    index);

            } else {
                const balanceOf = await vesta.methods.balanceOf(owner).call();

                const tokenIdsOfOwner = new Array(Number(balanceOf)).fill().map((_,
    index) =>
                    vesta.methods.tokenOfOwnerByIndex(owner, index).call()
                );

                tokenIds = await Promise.all(tokenIdsOfOwner);
            }


            const vestaPromise = tokenIds.map(tokenId =>
                getVestaData({ tokenId, vesta })
            );

            const go = await Promise.all(vestaPromise);

            setVestas(go);
            setLoading(false);
        } else {
            setLoading(false);
```

```
        }
    }, [vesta, owner, library?.utils]);

    useEffect(() => {
        update();
    }, [update])

    return {
        loading,
        allVestas,
        update,
    }
}

// Singular
const useVestaData = (tokenId = null) => {
    const [singleVesta, setVesta] = useState({});
    const [loading, setLoading] = useState(true);
    const vesta = useVesta();

    const update = useCallback(async () => {
        if (vesta && tokenId != null) {
            setLoading(true);

            const toSet = await getVestaData({ tokenId, vesta })
            setVesta(toSet);

            setLoading(false);
        }
    }, [vesta, tokenId])

    useEffect(() => {
        update();
    }, [update])

    return {
        loading,
        singleVesta,
        update,
    }

}


export { useVestasData, useVestaData };
```

Script C.11: *Hook* obtención de datos de un *NFT* en particular y lista de *NFTs* creados

C.8. Oráculo

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract PriceConsumerV3 {

    AggregatorV3Interface internal priceFeed;

    /**
     * Network: Goerli
     * Aggregator: ETH/USD
     * Address: 0xD4a33860578De61DBAbDc8BFdb98FD742fA7028e
     */
    constructor() {
        priceFeed = AggregatorV3Interface(0xD4a33860578De61DBAbDc8BFdb98FD742fA7028e)
    ;
    }

    /**
     * Returns the latest price
     */
    function getLatestPrice() public view returns (int) {
        (
            int price,
            uint timeStamp,
        ) = priceFeed.latestRoundData();
        return (price, timeStamp);
    }
}
```

Script C.12: (Solidity) lib/oraculos/PriceConsumerV3.sol

C.9. Verificaciones de seguridad en registro e inicio de sesión

```
const password = prompt("Ingrese su password")
if (password === undefined || password === null) {
  disconnect()
  return
}

if (password != null){
  if (password.length < 6){
    disconnect()
    alert("Password must contain 6 characters minimum")
    return
  } else if (
            password.search(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)./) == -1
        ){
    disconnect()
    alert(
            "Password must cotain lower case, upper case and number"
            )
    return
  }
}

const password_confirmation = prompt("Ingrese su password nuevamente")
if (password_confirmation === undefined ||
password_confirmation === null) {
  disconnect()
  return
}

if (password_confirmation !== password) {
  disconnect()
  alert("Passwords do not match")
  return
}

const first_name = prompt("Ingrese su nombre")
const last_name = prompt("Ingrese su apellido")
const telephone = prompt("Ingrese su telefono")


dispatch(auth.actions.startRegistration({
  registration: {
            email,
            password,
            password_confirmation,
            first_name,
            last_name,
            telephone
        },
  wallet: { account }
}))
```

C.10. Reducción de volumen

```
#USE venv/ python 3.10
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

data = pd.read_csv(FLAT_FILE_DIR)

string_to_num = {
    "'-9'": -9,
    "'-6'": -6,
    "'1'": 1,
    "'2'": 2,
    "'3'": 3,
    "'4'": 4,
    "'5'": 5,
    "'6'": 6,
    "'7'": 7,
    "'8'": 8,
    "'9'": 9,
    "'01'": 1,
    "'02'": 2,
    "'03'": 3,
    "'04'": 4,
    "'05'": 5,
    "'06'": 6,
    "'07'": 7,
    "'08'": 8,
    "'09'": 9,
}

final_dataframe = data[[
    'CONTROL',
    'CONDO',
    'YRBUILT',
    'STORIES',
    'FOUNDTYPE',
    'TOTROOMS',
    'BEDROOMS',
    'BATHROOMS',
    'UNITSIZE',
    'LOTSIZE',
    'PORCH',
    'FIREPLACE',
    'HEATTYPE',
    'HOTWATER',
    'WASHER',
    'DRYER',
    'KITCHSINK',
    'FRIDGE',
    'COOKTYPE',
    'DISHWASH',
    'LANDLINE',
    'CELLPHONE',
    'SOLAR',
    'WATSOURCE',
    'SEWTYPE',
    'NOWIRE',
    # 'PLUGS',
    'WALLCRACK',
    'FLOORHOLE',
    'ROACH',
    'RODENT',
    'NOWATFREQ',
```

```
    'LEAKI',
    'LEAKO',
    'ROOFHOLE',
    'WINBARS',
    'WALLSLOPE',
    'WALLSIDE',
    'MOLDKITCH',
    'MOLDBATH',
    'MOLDBEDRM',
    'MOLDLROOM',
    'MOLDBASEM',
    'MOLDOTHER',
    'HINCP',
    'MORTAMT',
    'RENT',
    'YRBUILT',
    'HOWBUY',
    'NEARABAND',
    'RATINGNH',
    'RATINGHS',
    'MARKETVAL',
    ]][data['INTSTATUS']== "'1'"]


final_dataframe = final_dataframe[final_dataframe.MARKETVAL>0]

# encode using https://pbpython.com/categorical-encoding.html

final_dataframe.dtypes
final_dataframe['more_3_bathrooms'] = np.where(final_dataframe["BATHROOMS"].str.
    replace(r'\'', '').astype(int) == 6, 1, 0)
final_dataframe['less_1_bathrooms'] = np.where(final_dataframe["BATHROOMS"].str.
    replace(r'\'', '').astype(int) > 6, 1, 0)
bathrooms_dict = {
    "'01'": 1,
    "'02'": 1.5,
    "'03'": 2,
    "'04'": 2.5,
    "'05'": 3,
    "'06'": 3,
    "'08'": 0,
    "'09'": 0,
    "'11'": 0,
    "'12'": 0,
    "'13'": 0
}

unitsize_dict = {
    "'1'": 'lt_500',
    "'2'": '500_to_749',
    "'3'": '750_to_999',
    "'4'": '1000_to_1499',
    "'5'": '1500_to_1999',
    "'6'": '2000_to_2499',
    "'7'": '2500_to_2999',
    "'8'": '3000_to_3999',
    "'9'": '4000 or more',
    "'-9'": 'not_reported'
 }

cleanup_dict = {
    'BATHROOMS': bathrooms_dict ,
    'UNITSIZE': unitsize_dict,
    'LOTSIZE': string_to_num,
}
final_dataframe = final_dataframe.replace(cleanup_dict)

final_dataframe = pd.get_dummies(final_dataframe, columns=['UNITSIZE'], prefix=['
```

```
    square_footage'])

condo_dict = {
    "'1'": 1, # condominium
    "'2'": 0, # no condominium
}

porch_dict = {
    "'1'": 1, # porch
    "'2'": 0, # no porch
}

washer_dict = {
    "'1'": 1, # washer
    "'2'": 0, # no washer
}

dryer_dict = {
    "'1'": 1, # dryer
    "'2'": 1, # dryer
    "'3'": 1, # dryer
    "'4'": 1, # dryer
    "'5'": 0, # no dryer
}

kitchen_sink_dict = {
    "'1'": 1, # sink
    "'2'": 0, # no sink
}

fridge_dict = {
    "'1'": 1, # fridge
    "'2'": 0, # no fridge
}

dishwasher_dict = {
    "'1'": 1, # dishwasher
    "'2'": 0, # no dishwasher
}

cooktype_dict = {
"'1'": 'cooking_stove_or_range',
"'2'": 'burners_no_stove_or_range',
"'3'": 'microwave_oven_only',
"'4'": 'no_kitchen_facilities'
}

fireplace_dict = {
"'1'": 1,
"'2'": 1,
"'3'": 1,
"'4'": 0
}

landline_dict = {
"'-6'": 0,
"'-9'": 0,
"'1'": 1,
"'2'": 0
}

solar_dict = {
    "'1'": 'has_solar',
    "'2'": 'no_has_solar',
    "'-9'": 'not_reported',
}

watsource_dict = {
```

```
 "'1'": 'public_or_private_system',
 "'2'": 'individual_well',
 "'3'": 'other'
}

heattype_dict = {
"'01'": 'warm-air_furnace',
"'02'": 'steam_or_hot_water_system',
"'03'": 'electric_heat_pump',
"'04'": 'built-in_electric_units',
"'05'": 'floor,_wall,_or_other_built-in_hot-air_units_without_ducts',
"'06'": 'room_heaters_with_flue',
"'07'": 'room_heaters_without_flue',
"'08'": 'portable_electric_heaters',
"'09'": 'stoves',
"'10'": 'fireplaces_with_inserts',
"'11'": 'fireplaces_without_inserts',
"'14'": 'cooking_stove',
"'12'": 'other',
"'13'": 'none',
"'14'": 'none',
}

foundtype_dict = {
"'1'": 'basement_under_all_of_house',
"'2'": 'basement_under_part_of_house',
"'3'": 'crawl_space',
"'4'": 'concrete_slab',
"'5'": 'mobile_home_set_on_masonry_foundation',
"'6'": 'mobile_home_resting_on_concrete_pad',
"'7'": 'mobile_home_up_on_blocks,_but_not_on_concrete_pad',
"'9'": 'mobile_home_foundation_not_reported',
"'8'": 'foundation_setup_in_some_other_way',
}

hotwater_dict = {
"'1'": 'electricity',
"'2'": 'piped_gas',
"'3'": 'bottled_gas',
"'4'": 'fuel_oil',
"'5'": 'solar_energy',
"'6'": 'other',
"'7'": 'no_hot_piped_water',
}

sewtype_dict = {
    "'-9'": 'not_reported',
     "'01'": 'public',
     "'02'": 'septic_tank',
     "'03'": 'pump',
     "'04'": 'above_natural_soil_water',
     "'05'": 'treated_wastedwater',
     "'06'": 'stank_or_cesspool_other',
     "'07'": 'other',
     "'08'": 'other',
     "'09'": 'other',
     "'10'": 'none'
}

nowire_dict = {
    "'1'": 'with_electrical_wiring',
    "'2'": 'exposed_wiring',
    "'3'": 'no_electrical_wiring'
}

wallcrack_dict = {
    "'1'": 1, # wallcrack
    "'2'": 0, # no wallcrack
```

```
}

floorhole_dict = {
    "'1'": 1, # floorhole
    "'2'": 0, # no floorhole
}

roach_dict = {
    "'-6'": 'not_reported',
    "'1'": 'roach_last_12_months',
    "'2'": 'roach_last_12_months',
    "'3'": 'roach_last_12_months',
    "'4'": 'roach_last_12_months',
    "'5'": 'no_roach_last_12_months',
}

rodent_dict = {
    "'-6'": 'not_reported',
    "'1'": 'rodent_last_12_months',
    "'2'": 'rodent_last_12_months',
    "'3'": 'rodent_last_12_months',
    "'4'": 'rodent_last_12_months',
    "'5'": 'no_rodent_last_12_months',
}

nowatfreq_dict = {
"'-6'": 'no_wat_reported',
"'-9'": 'no_water_stoppage_reported',
"'0'":  'no_water_stoppage_reported',
"'1'": '1_water_stoppage',
"'2'": '2_water_stoppage',
"'3'": '3_water_stoppage',
"'4'": '4_or_more_water_stoppage',
"'5'": '4_or_more_water_stoppage',
"'6'": '4_or_more_water_stoppage',
"'7'": '4_or_more_water_stoppage',
"'8'": '4_or_more_water_stoppage',
}

leaki_dict = {
    "'1'": 'with_leakage_inside_structure',
    "'2'": 'without_leakage_inside_structure',
    "'-6'": 'not_reported',
}

leako_dict = {
    "'1'": 'with_leakage_outside_structure',
    "'2'": 'without_leakage_outside_structure',
    "'-6'": 'not_reported',
}

roofhole_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
"'1'": 'hole_in_roof',
"'2'": 'no_hole_in_roof',
}

winbars_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
"'1'": 'bars_on_windows',
"'2'": 'no_bars_on_windows',
}

wallslope_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
```

```
"'1'": 'sloping_outside_walls',
"'2'": 'no_sloping_outside_walls',
}

wallside_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
"'1'": 'missing_outside_wall_material',
"'2'": 'not_missing_outside_wall_material',
}

mold_dict = {
    "'-6'": 'not_reported',
    "'-9'": 'not_reported',
    "'1'": 'with_mold',
    "'2'": 'with_no_mold',
}

howbuy_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
"'1'": 'already_built',
"'2'": 'sales_agreement',
"'3'": 'contractor',
"'4'": 'built_it_yourself',
"'5'": 'inheritance_or_gift',
}

nearaband_dict = {
"'-6'": 'not_reported',
"'-9'": 'not_reported',
"'1'": '1_building',
"'2'": 'more_1_building',
"'3'": 'none',
"'4'": 'no_buildings',
}

final_dataframe = final_dataframe.replace({
    'CONDO': condo_dict,
    'FOUNDTYPE': foundtype_dict,
    'PORCH': porch_dict,
    'FIREPLACE': fireplace_dict,
    'HEATTYPE': heattype_dict,
    'HOTWATER': hotwater_dict,
    'WASHER': washer_dict,
    'DRYER': dryer_dict,
    'KITCHSINK': kitchen_sink_dict,
    'FRIDGE': fridge_dict,
    'COOKTYPE': cooktype_dict,
    'DISHWASH': dishwasher_dict,
    'LANDLINE': landline_dict,
    'SOLAR': solar_dict,
    'WATSOURCE': watsource_dict,
    'SEWTYPE': sewtype_dict,
    'NOWIRE': nowire_dict,
    'WALLCRACK': wallcrack_dict,
    'FLOORHOLE': floorhole_dict,
    'ROACH': roach_dict,
    'RODENT': rodent_dict,
    'NOWATFREQ': nowatfreq_dict,
    'LEAKI': leaki_dict,
    'LEAKO': leako_dict,
    'ROOFHOLE': roofhole_dict,
    'WALLSLOPE': wallslope_dict,
    'WALLSIDE': wallside_dict,
    'MOLDKITCH': mold_dict,
    'MOLDBATH': mold_dict,
    'MOLDBEDRM': mold_dict,
```

```
    'MOLDLROOM': mold_dict,
    'MOLDBASEM': mold_dict,
    'MOLDOTHER': mold_dict,
    'HOWBUY': howbuy_dict,
    'NEARABAND': nearaband_dict,
})

final_dataframe = pd.get_dummies(final_dataframe, columns=[
    'COOKTYPE',
    'SOLAR',
    'WATSOURCE',
    'HEATTYPE',
    'FOUNDTYPE',
    'HOTWATER',
    'SEWTYPE',
    'NOWIRE',
    'WALLCRACK',
    'FLOORHOLE',
    'RODENT',
    'NOWATFREQ',
    'LEAKI',
    'LEAKO',
    'ROOFHOLE',
    'WINBARS',
    'WALLSLOPE',
    'WALLSIDE',
    'MOLDKITCH',
    'MOLDBATH',
    'MOLDBEDRM',
    'MOLDLROOM',
    'MOLDBASEM',
    'MOLDOTHER',
    'HOWBUY',
    'NEARABAND',
    'ROACH'
], prefix=[
    'cook_type',
    'solar',
    'water_source',
    'heat_type',
    'foundation_type',
    'hot_water',
    'sewage_type',
    'no_wire',
    'wall_crack',
    'floor_hole',
    'rodent',
    'no_water_frequency',
    'leak_inside',
    'leak_outside',
    'roof_hole',
    'window_bars',
    'wall_slope',
    'wall_side',
    'mold_kitchen',
    'mold_bathroom',
    'mold_bedroom',
    'mold_living_room',
    'mold_basement',
    'mold_other',
    'how_buy',
    'near_abandoned',
    'roach'
])
```

ANEXO D

Simulaciones de transacciones

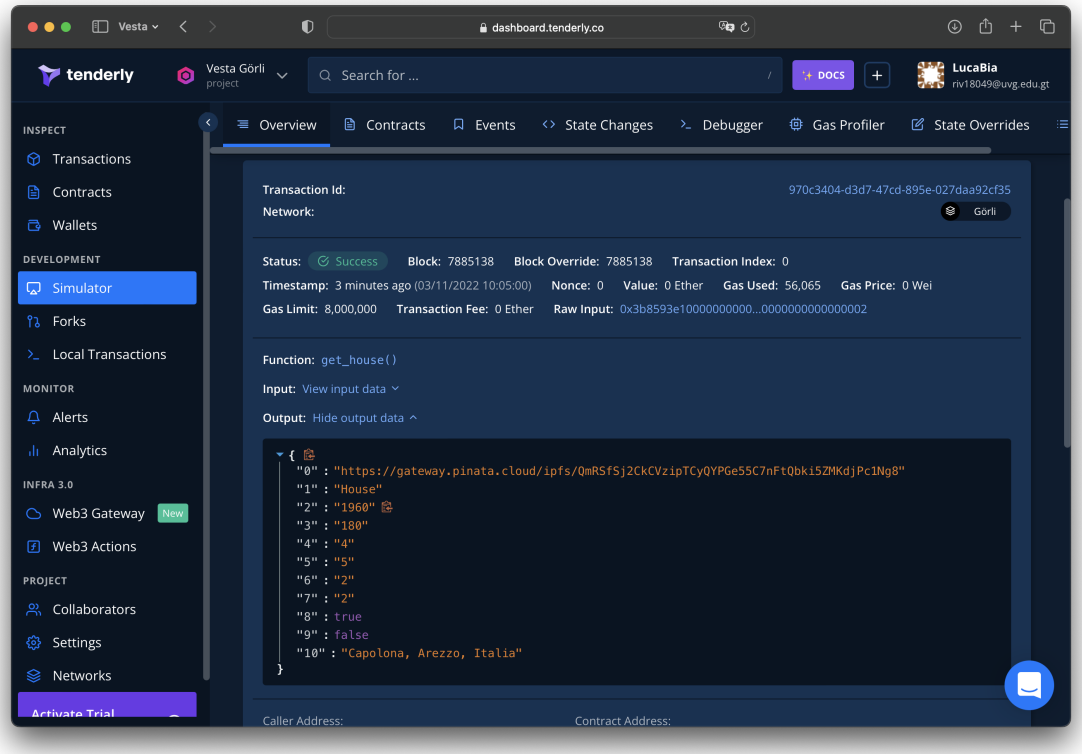


Figura D.1: Simulación de transacción exitosa

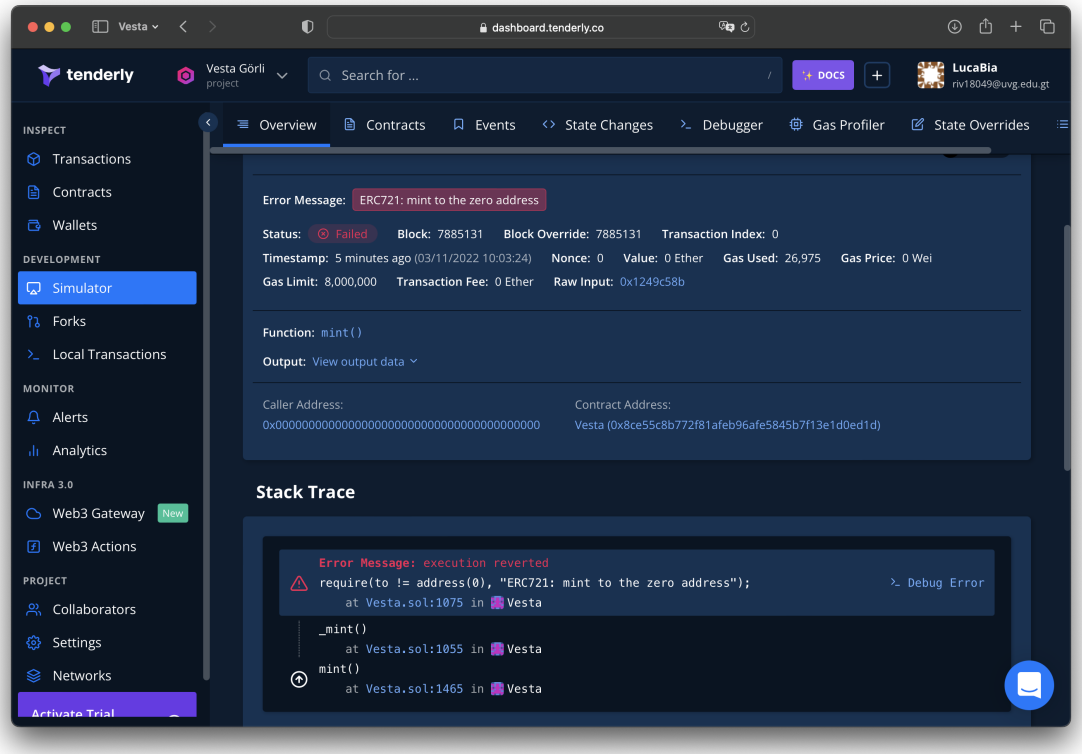


Figura D.2: Simulación de transacción fallida

ANEXO E

Despliegue de *smart contract* en diferente redes

- Ethereum *mainnet*:
  - https://etherscan.io/address/0x0e2e19ad47dda0b8a696824d24de374c968ba28e
- Ethereum Görli *testnet*:
  - https://goerli.etherscan.io/address/0x8Ce55c8B772F81AfEb96afe5845B7f13e1d0ed1d
- Avalanche Fuji *testnet*:
  - https://testnet.snowtrace.io/address/0x6dC99dfaecdF96Db0944015EF6d43DEb77eAA595

ANEXO F

Implementación de *backend*

```
      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  .gitattributes
      create  Gemfile
         run  git init from "."
Initialized empty Git repository in /Users/willi/Desktop/repos/temp/realestate-nft-marketplace/.git/
      create  app
      create  app/assets/config/manifest.js
      create  app/assets/stylesheets/application.css
      create  app/channels/application_cable/channel.rb
      create  app/channels/application_cable/connection.rb
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/jobs/application_job.rb
      create  app/mailers/application_mailer.rb
      create  app/models/application_record.rb
      create  app/views/layouts/application.html.erb
      create  app/views/layouts/mailer.html.erb
      create  app/views/layouts/mailer.text.erb
      create  app/assets/images
      create  app/assets/images/.keep
      create  app/controllers/concerns/.keep
      create  app/models/concerns/.keep
      create  bin
      create  bin/rails
      create  bin/rake
      create  bin/setup
      create  config
      create  config/routes.rb
```

Figura F.1: Ejecución del comando rails *new*.

```
(base) ➜  realestate-nft-marketplace git:(main) ✗ rails generate scaffold User
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bundler is installing for. Bundler
e lock --add-platform x86-mingw32 x86-mswin32 x64-mingw32 java`.
      invoke  active_record
      create    db/migrate/20221103050724_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
      invoke  resource_route
       route    resources :users
      invoke  scaffold_controller
      create    app/controllers/users_controller.rb
      invoke    erb
      create      app/views/users
      create      app/views/users/index.html.erb
      create      app/views/users/edit.html.erb
      create      app/views/users/show.html.erb
      create      app/views/users/new.html.erb
      create      app/views/users/_form.html.erb
      create      app/views/users/_user.html.erb
      invoke    resource_route
      invoke    test_unit
      create      test/controllers/users_controller_test.rb
      create      test/system/users_test.rb
      invoke    helper
      create      app/helpers/users_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/users/index.json.jbuilder
      create      app/views/users/show.json.jbuilder
      create      app/views/users/_user.json.jbuilder
(base) ➜  realestate-nft-marketplace git:(main) ✗
```

Figura F.2: Ejecución del comando rails *generate scaffold*.

```
Created database 'nft_realestate_dev'
Database 'nft_realestate_dev' already exists
== 20220901052335 CreateUsers: migrating ======================================
-- create_table(:users)
   -> 0.0105s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0033s
-- add_index(:users, :reset_password_token, {:unique=>true})
   -> 0.0016s
== 20220901052335 CreateUsers: migrated (0.0155s) =============================

== 20221026233848 CreateProperties: migrating =================================
-- create_table(:properties)
   -> 0.0068s
-- add_reference(:properties, :user, {:foreign_key=>true})
   -> 0.0084s
== 20221026233848 CreateProperties: migrated (0.0152s) ========================

== 20221029073239 CreateUserWallets: migrating ================================
-- create_table(:user_wallets)
   -> 0.0052s
-- add_reference(:user_wallets, :user, {:foreign_key=>true})
   -> 0.0043s
== 20221029073239 CreateUserWallets: migrated (0.0096s) =======================

== 20221030011627 AlterUser1: migrating =======================================
-- add_column(:users, :first_name, :string)
   -> 0.0018s
-- add_column(:users, :last_name, :string)
   -> 0.0009s
-- add_column(:users, :telephone, :string)
```

Figura F.3: Ejecución del comando rails db:migrate.

ANEXO G

Server

G.1. Instancia

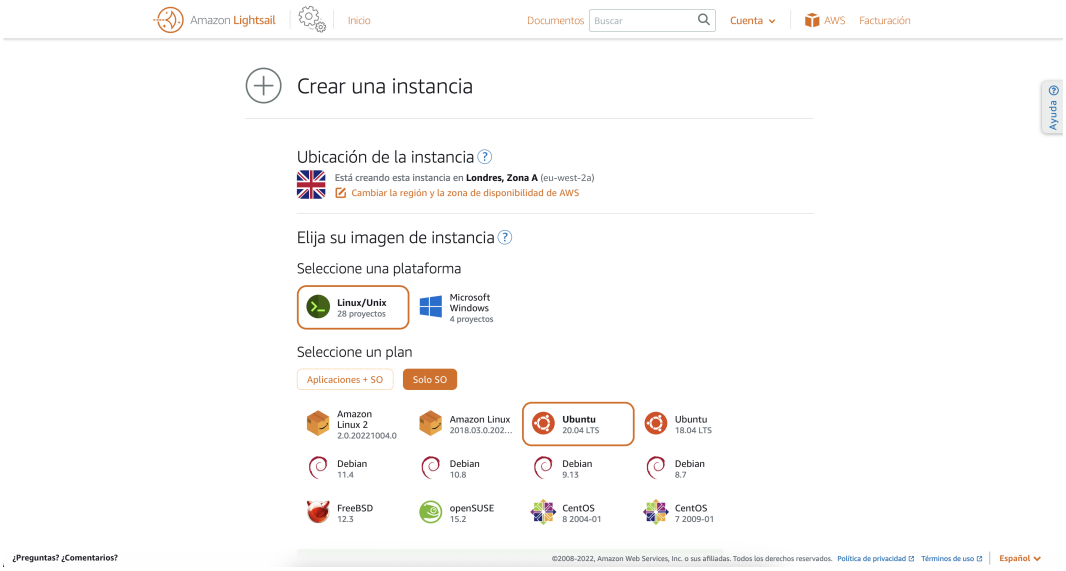


Figura G.1: Creación de instancia 1.

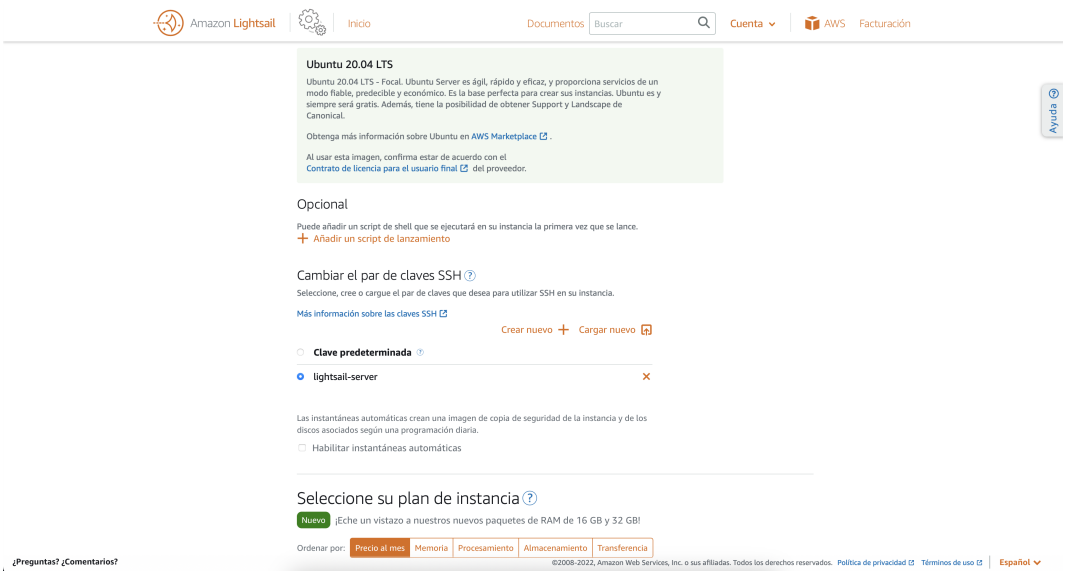


Figura G.2: Creación de instancia 2.

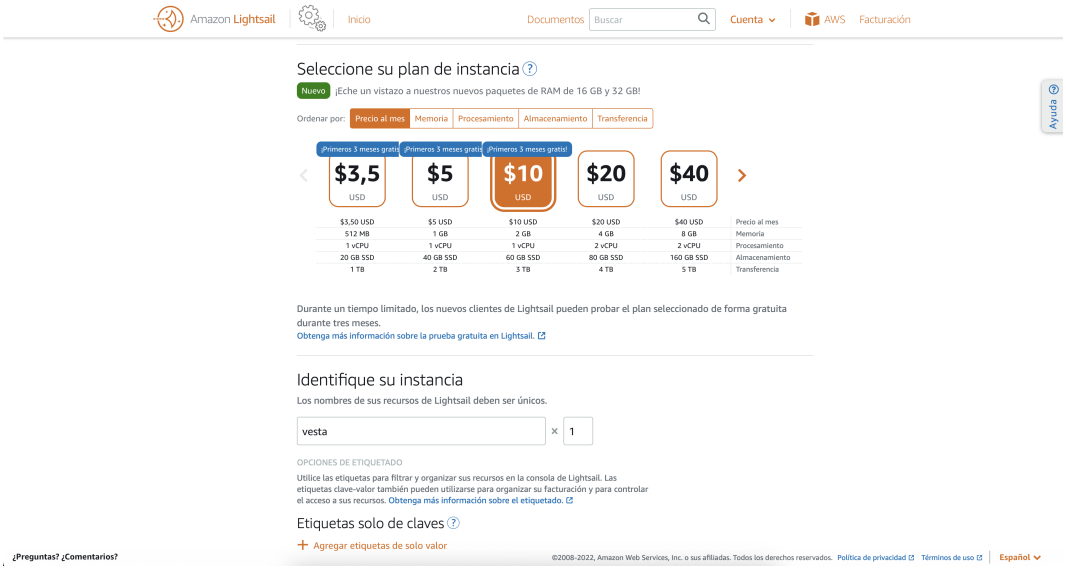


Figura G.3: Creación de instancia 3.

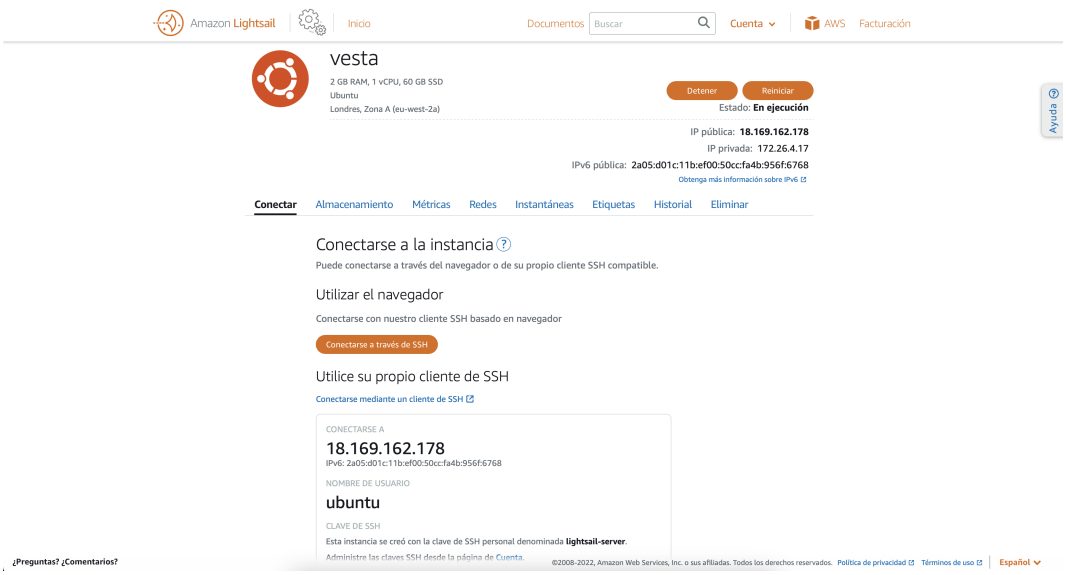


Figura G.4: Instancia.

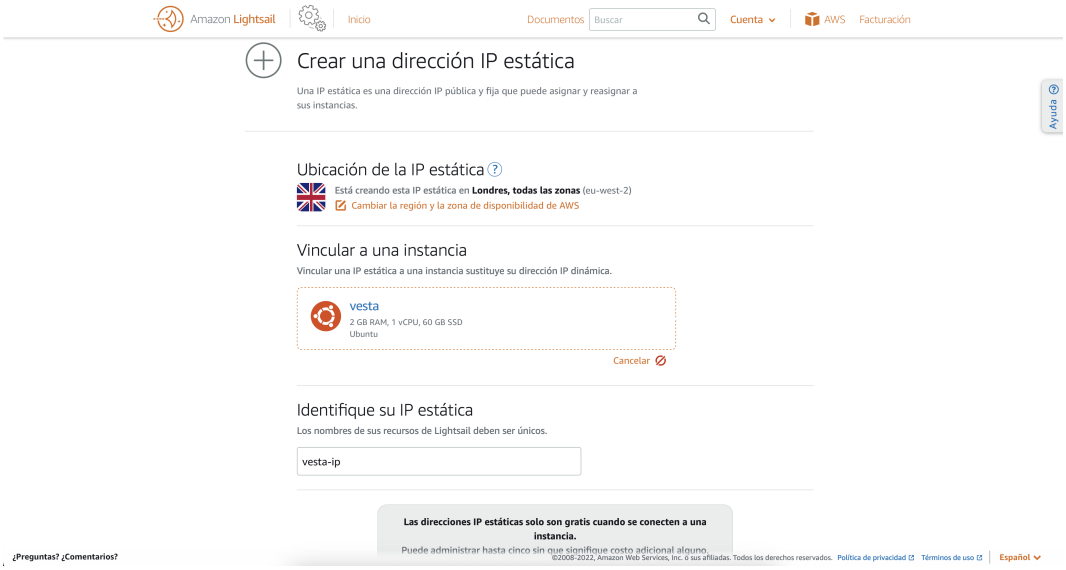


Figura G.5: Creación de IP estática.

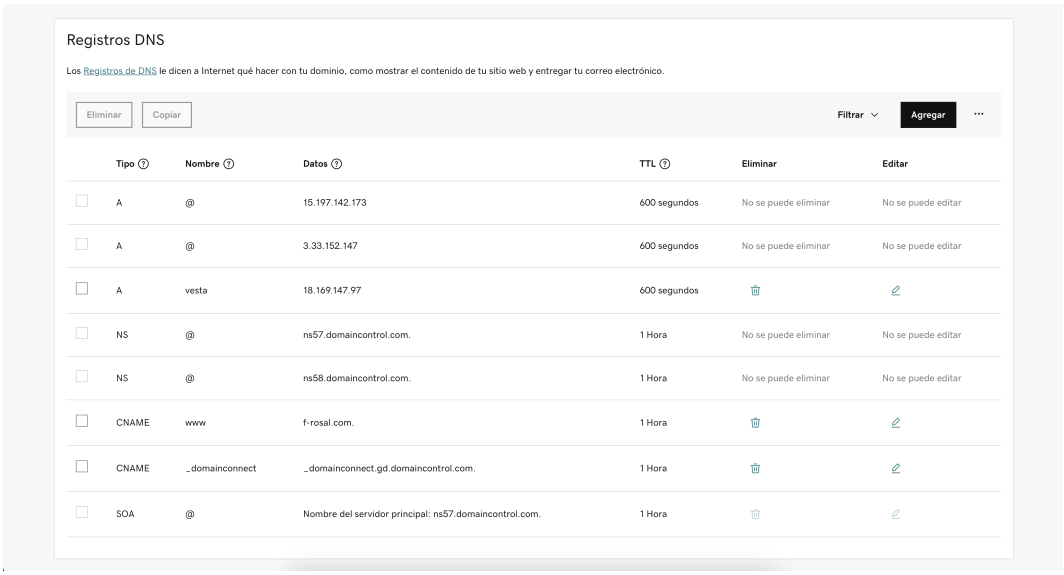


Figura G.6: Configuración de registros DNS.

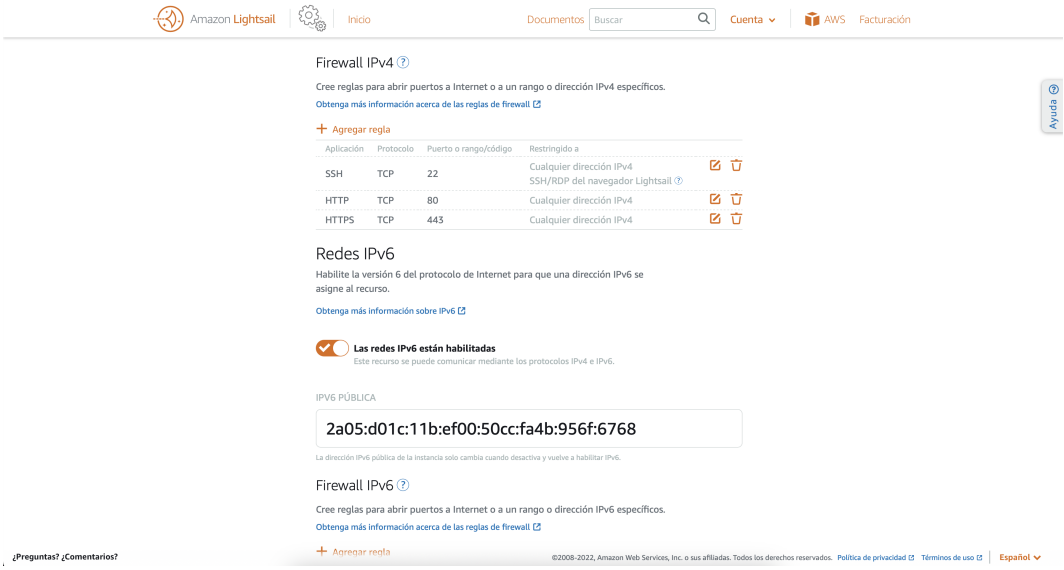


Figura G.7: Configuración de puertos.

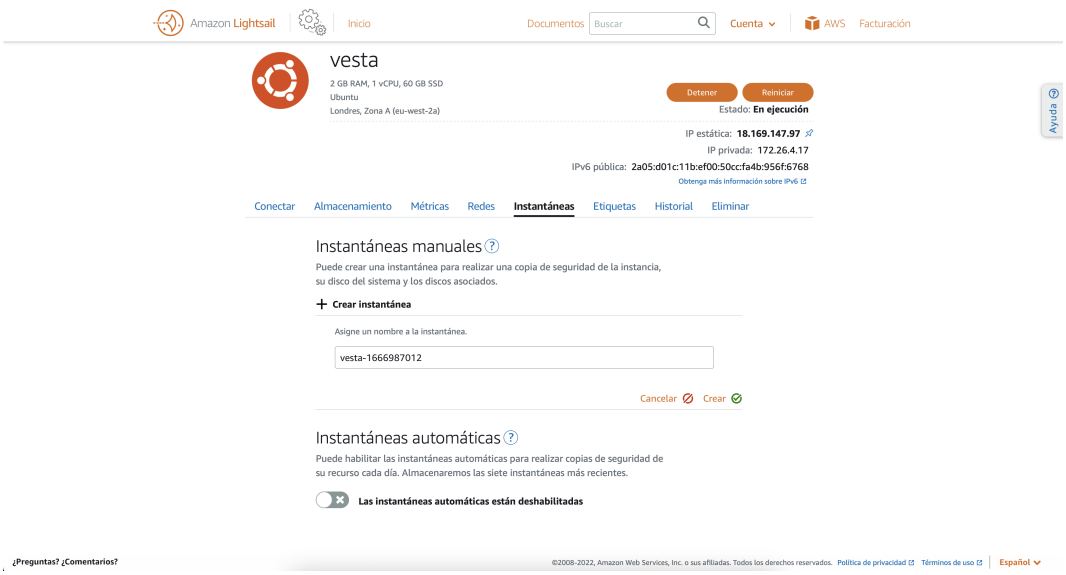


Figura G.8: Creación de *snapshot* 1.

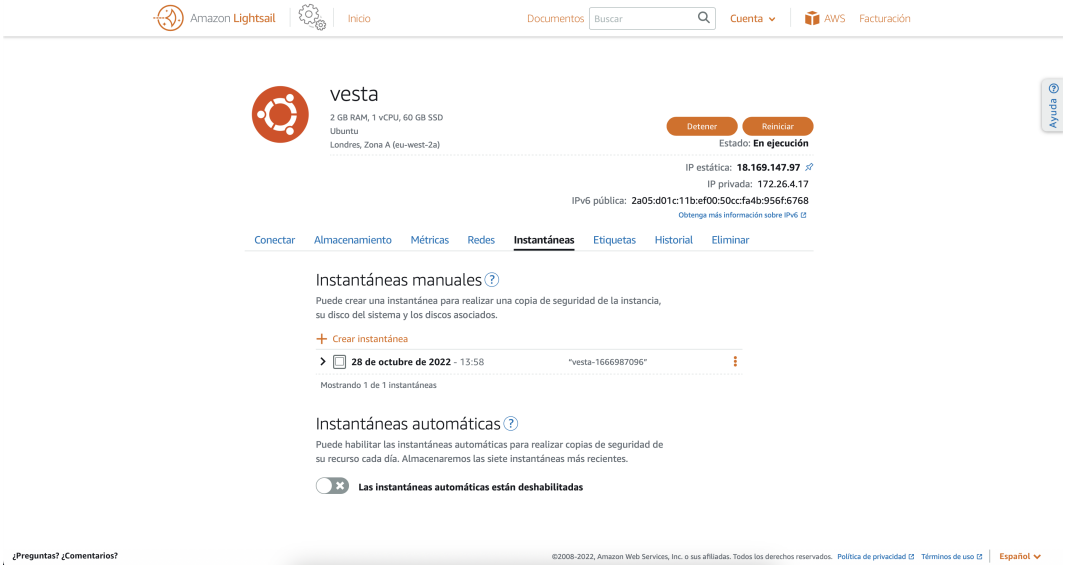


Figura G.9: Creación de *snapshot* 2.

G.2. DB

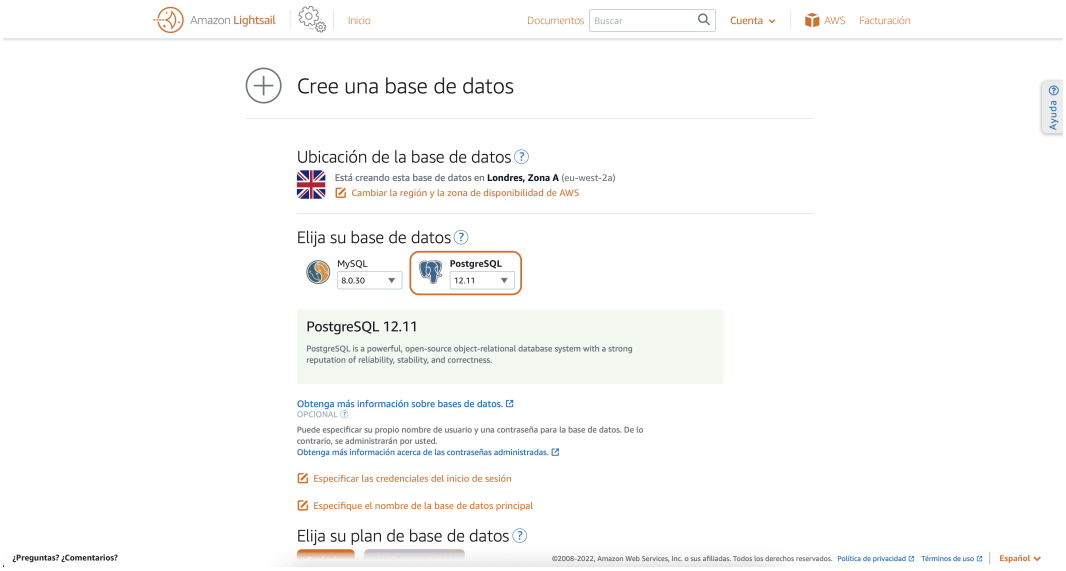


Figura G.10: Creación de base de datos 1.

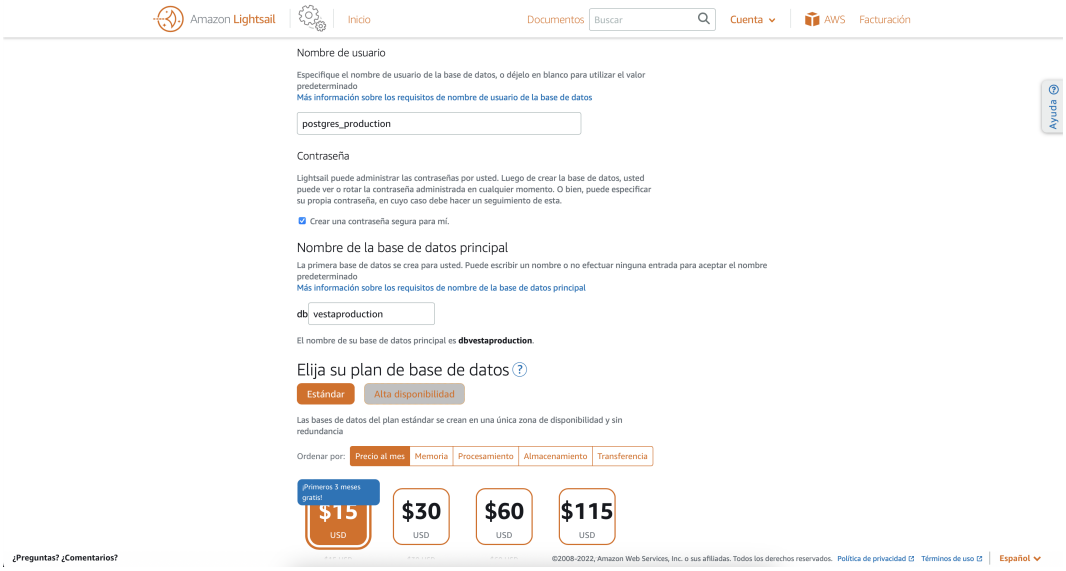


Figura G.11: Creación de base de datos 2.

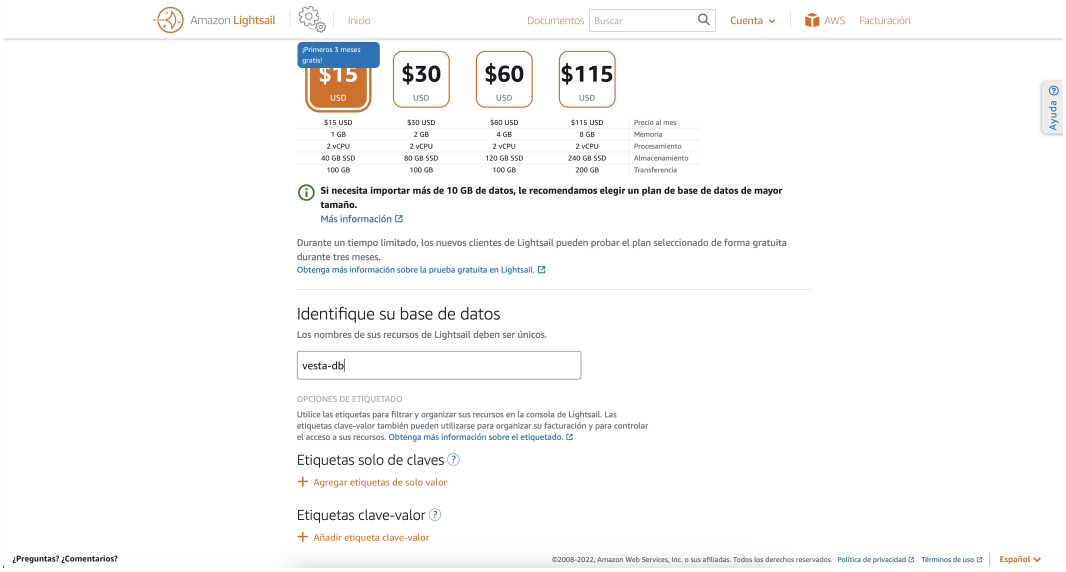


Figura G.12: Creación de base de datos 3.

G.3. Almacenamiento

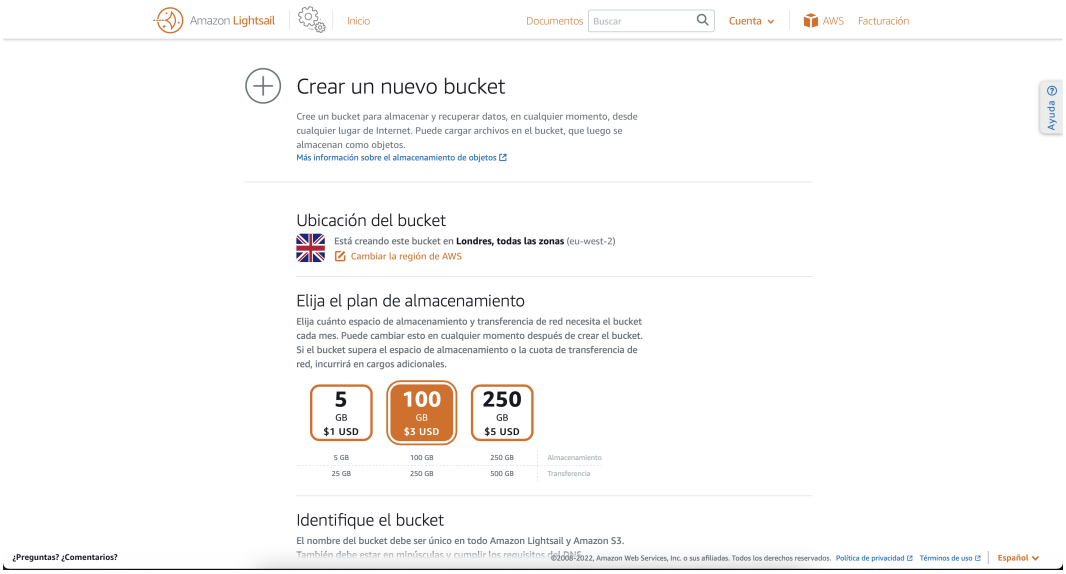


Figura G.13: Creación de almacenamiento 1.

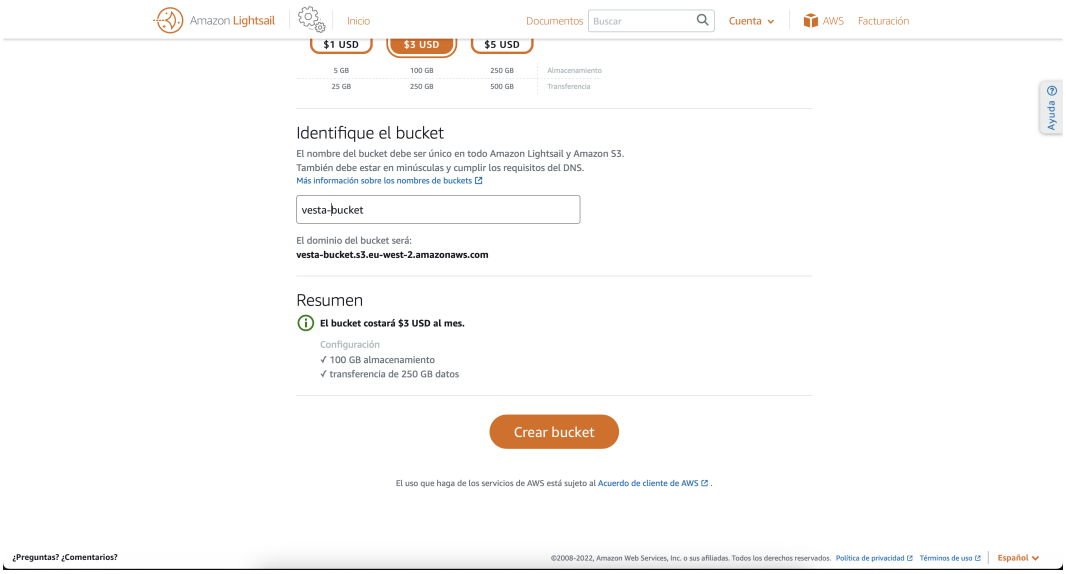


Figura G.14: Creación de almacenamiento 2.

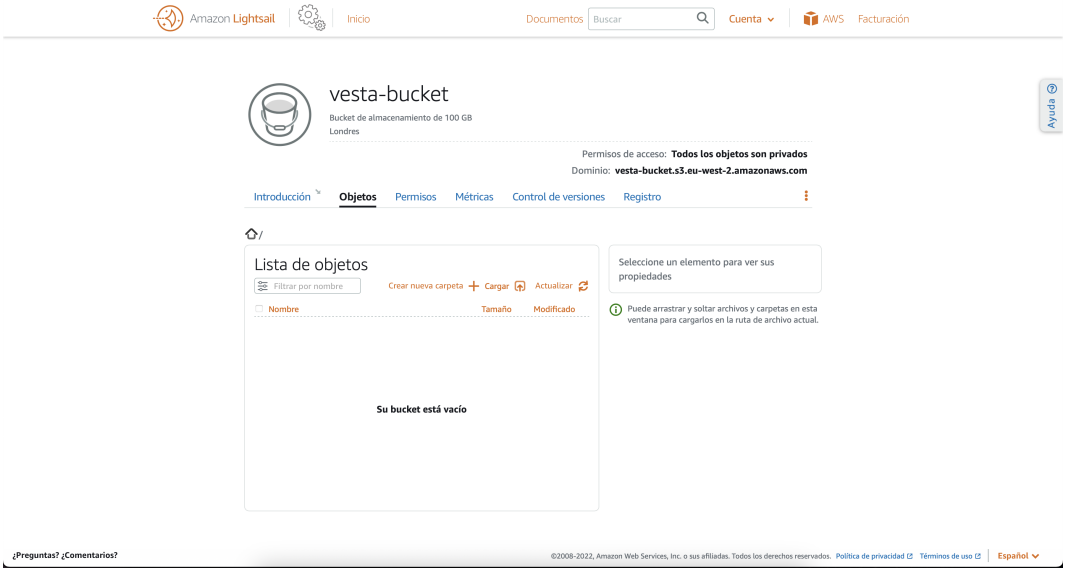


Figura G.15: Creación de almacenamiento 3.

G.4. *Server*

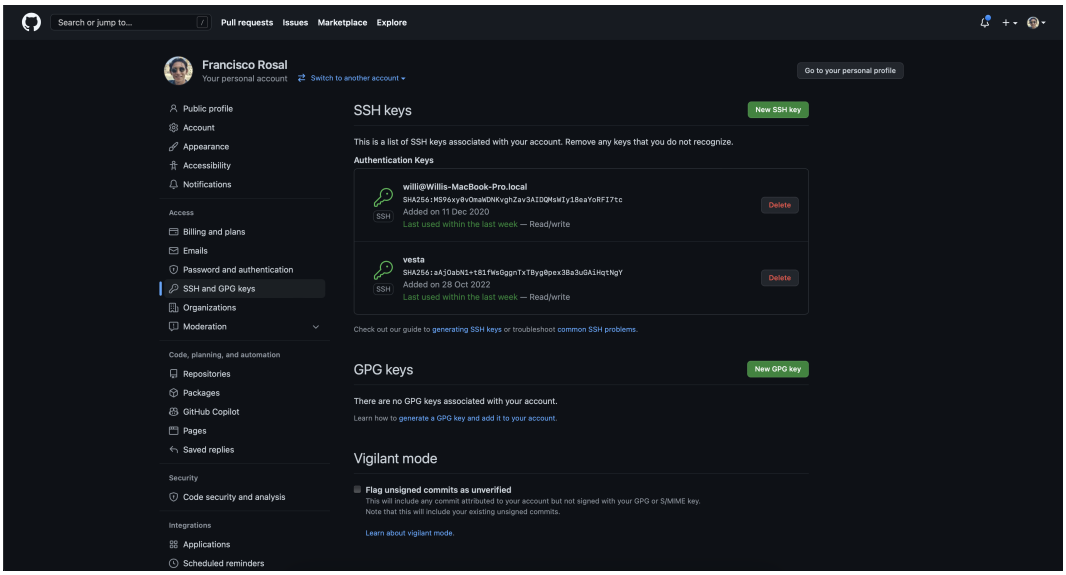


Figura G.16: Configuración de llaves SSH para el servidor

Figura G.17: Configuración de nginx en el servidor

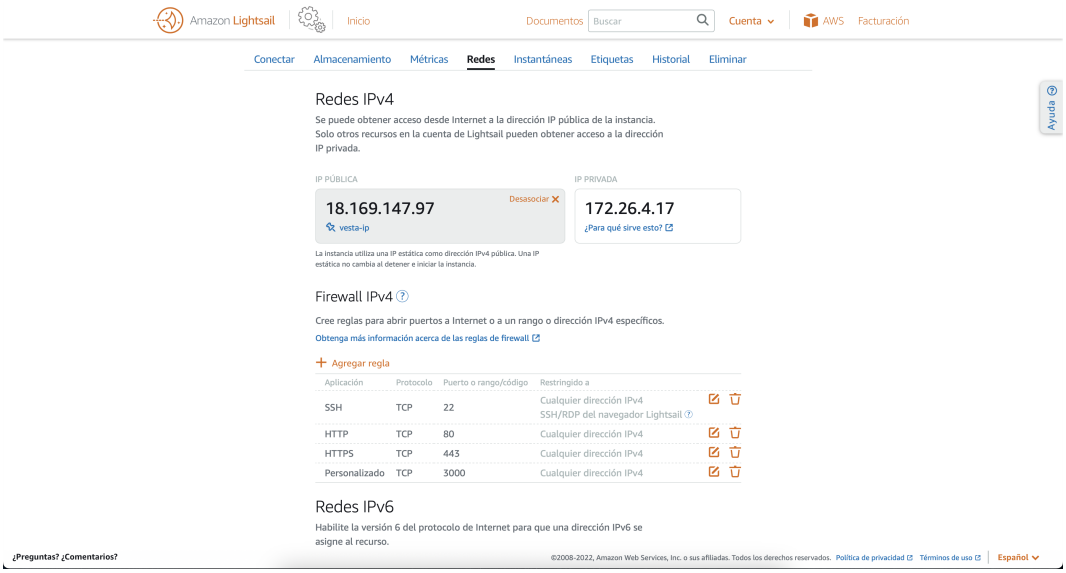


Figura G.18: Adición de puerto para *backend*

ANEXO H

Listado de columnas a utilizar para predecir el precio de una propiedad

- CONDO
- YRBUILT
- STORIES
- TOTROOMS
- BEDROOMS
- BATHROOMS
- LOTSIZE
- PORCH
- FIREPLACE
- WASHER
- DRYER
- KITCHSINK
- FRIDGE
- DISHWASH
- LANDLINE
- CELLPHONE
- HINCP
- MORTAMT
- RENT

- YRBUILT
- RATINGNH
- RATINGHS
- MARKETVAL
- more_3_bathrooms
- less_1_bathrooms
- square_footage_1000_to_1499
- square_footage_1500_to_1999
- square_footage_2000_to_2499
- square_footage_2500_to_2999
- square_footage_3000_to_3999
- square_footage_4000 or more
- square_footage_500_to_749
- square_footage_750_to_999
- square_footage_lt_500
- square_footage_not_reported
- cook_type_burners_no_stove_or_range
- cook_type_cooking_stove_or_range
- cook_type_microwave_oven_only
- cook_type_no_kitchen_facilities
- solar_has_solar
- solar_no_has_solar
- solar_not_reported
- water_source_individual_well
- water_source_other
- water_source_public_or_private_system
- heat_type_built-in_electric_units
- heat_type_electric_heat_pump
- heat_type_fireplaces_with_inserts
- heat_type_fireplaces_without_inserts
- heat_type_floor,_wall,_or_other_built-in_hot-air_units_without_ducts
- heat_type_none
- heat_type_other

- heat_type_portable_electric_heaters
- heat_type_room_heaters_with_flue
- heat_type_room_heaters_without_flue
- heat_type_steam_or_hot_water_system
- heat_type_stoves
- heat_type_warm-air_furnace
- foundation_type_'-6'
- foundation_type_basement_under_all_of_house
- foundation_type_basement_under_part_of_house
- foundation_type_concrete_slab
- foundation_type_crawl_space
- foundation_type_foundation_setup_in_some_other_way
- foundation_type_mobile_home_foundation_not_reported
- foundation_type_mobile_home_resting_on_concrete_pad
- foundation_type_mobile_home_set_on_masonry_foundation
- foundation_type_mobile_home_up_on_blocks,_but_not_on_concrete_pad
- hot_water_bottled_gas
- hot_water_electricity
- hot_water_fuel_oil
- hot_water_no_hot_piped_water
- hot_water_other
- hot_water_piped_gas
- hot_water_solar_energy
- sewage_type_above_natural_soil_water
- sewage_type_none
- sewage_type_not_reported
- sewage_type_other
- sewage_type_public
- sewage_type_pump
- sewage_type_septic_tank
- sewage_type_stank_or_cesspool_other
- sewage_type_treated_wastedwater
- no_wire_exposed_wiring

- no_wire_no_electrical_wiring
- no_wire_with_electrical_wiring
- wall_crack_0
- wall_crack_1
- floor_hole_0
- floor_hole_1
- rodent_no_rodent_last_12_months
- rodent_rodent_last_12_months
- no_water_frequency_1_water_stoppage
- no_water_frequency_2_water_stoppage
- no_water_frequency_3_water_stoppage
- no_water_frequency_4_or_more_water_stoppage
- no_water_frequency_no_wat_reported
- no_water_frequency_no_water_stoppage_reported
- leak_inside_with_leakage_inside_structure
- leak_inside_without_leakage_inside_structure
- leak_outside_with_leakage_outside_structure
- leak_outside_without_leakage_outside_structure
- roof_hole_hole_in_roof
- roof_hole_no_hole_in_roof
- roof_hole_not_reported
- window_bars_'-6'
- window_bars_'-9'
- window_bars_'1'
- window_bars_'2'
- wall_slope_no_sloping_outside_walls
- wall_slope_not_reported
- wall_slope_sloping_outside_walls
- wall_side_missing_outside_wall_material
- wall_side_not_missing_outside_wall_material
- wall_side_not_reported
- mold_kitchen_not_reported
- mold_kitchen_with_mold

- mold_kitchen_with_no_mold
- mold_bathroom_not_reported
- mold_bathroom_with_mold
- mold_bathroom_with_no_mold
- mold_bedroom_not_reported
- mold_bedroom_with_mold
- mold_bedroom_with_no_mold
- mold_living_room_not_reported
- mold_living_room_with_mold
- mold_living_room_with_no_mold
- mold_basement_not_reported
- mold_basement_with_mold
- mold_basement_with_no_mold
- mold_other_not_reported
- mold_other_with_mold
- mold_other_with_no_mold
- how_buy_already_built
- how_buy_built_it_yourself
- how_buy_contractor
- how_buy_inheritance_or_gift
- how_buy_not_reported
- how_buy_sales_agreement
- near_abandoned_1_building
- near_abandoned_more_1_building
- near_abandoned_no_buildings
- near_abandoned_none
- near_abandoned_not_reported
- roach_no_roach_last_12_months
- roach_roach_last_12_months

ANEXO I

Repositorio del proyecto

El repositorio del proyecto se puede encontrar en https://github.com/UVG-Teams/realestate-nft-marketplace.

ANEXO J

Glosario

Glosario

***Artifact*:** Es un archivo JSON que contiene toda la información del resultado de la compilación de un *smart contract*. 1, 58

***Backend*:** Término utilizado para referirse al área lógica de una aplicación web. 1, 84

***Blockchain*:** *Blockchain* es una red de almacenamiento descentralizado que representa y funciona igual a un libro mayor compartido. 1

***Branch*:** Una rama en Git es simplemente un puntero ligero a un *commit*. 1

***Commit*:** Acción de guardar el estado actual de los archivos en el historial del control de versiones. 1

***Crackear*:** Actividades que comprometen un sistema o dispositivo para realizar fines maliciosos. 1

***Dataset*:** Cualquier grupo de registros con nombre se denomina *dataset*. Los *datasets* pueden contener información, como registros médicos o registros de seguros, para que los utilice un programa que se ejecuta en el sistema. También se utilizan para almacenar información que necesitan las aplicaciones o el propio sistema operativo, como programas fuente, bibliotecas de macros o variables o parámetros del sistema [47]. 1

***Debuggear*:** Acción de buscar dónde ocurre un error en código. 1

***Debuggeo*:** Proceso de encontrar la causa de un problema y solucionarlo en un sistema informático o de programación. 1, 120

***Form-data*:** La interfaz *FormData* proporciona una forma de construir fácilmente un conjunto de pares clave/valor que representan campos de formulario y sus valores [71]. 1, 86

***Framework*:** En los sistemas informáticos, un *framework* suele ser una estructura en capas que indica qué tipo de programas pueden o deben construirse y cómo se interrelacionarían. Algunos también incluyen programas reales, especifican interfaces de programación u ofrecen herramientas de programación para usar los marcos [65]. 1, 84

***Frontend*:** Es la parte del desarrollo web que se encarga de la interacción con los usuarios a través del diseño de pantallas y mostrar o pedir la información. 1

***Hackear*:** Actividades realizadas para comprometer dispositivos o sistemas y poder así obtener acceso a información. 1

***Header*:** Información complementaria que contiene datos importantes sobre como tratar el bloque de información a ejecutar. 1

***Merge*:** Acción de fusionar dos ramas en git. 1

***NFT*:** Acrónimo para *Non-Fungible Token*, o Token No Fungible en español. Los tokens son unidades de valor que se le asignan a un modelo de negocio, como por ejemplo el de las criptomonedas [33]. 1

***NFT*:** Abreviación de *Non Fungible Token* o Token No Fungible. 1

***Non-Fungible Token*:** Un token no fungible es un activo digital encriptado que representa algo único. 1

***Payload*:** El *payload* son un conjunto de datos transmitidos que facilitan la entrega de información. 1

***Phishing*:** Conjunto de técnicas con las cuales un atacante se hace pasar por una empresa, persona o servicio para ganar la confianza de la persona y así poder manipularla. 1

***Query*:** Solicitud a base de datos. 1

***Scope*:** Alcance de una entidad o variable. 1

***Script*:** Es una secuencia de comandos. 1

***Smart contract*:** Un "contrato inteligente" (*smart contract* en inglés) es un programa que se ejecuta en la cadena de bloques. Es una colección de código (sus funciones) y datos (su estado) que reside en una dirección específica en la cadena de bloques [44]. 1

***Software*:** Sistema informático. 1

***Stateless*:** Significa que no hay registro de interacciones anteriores y cada solicitud de interacción debe manejarse en función de la información que viene con ella. 1, 84

***Testing*:** Acción de realizar *tests* automatizados en una aplicación. 1

***Web3*:** Es una idea para una iteración nueva de la World Wide Web basada en tecnología *blockchain*. 1

ABI: *Contract Aplication Binary Interface*, es la forma estándar de interactuar con contratos en el ecosistema de Ethereum, tanto fuera de blockchain como dentro (de contrato a contrato [28]. 1, 58

Amenaza: Peligro potencial hacia algún sistema o su información. 1

API: Abreviación de *Application Programming Interface* o Interfaz de Programación de Aplicaciones. 1

BSD: BSD significa "Distribución de software de Berkeley" (*Berkley Software Distribution* en inglés). Es el nombre de las distribuciones de código fuente de la Universidad de California, Berkeley, que originalmente eran extensiones del sistema operativo Research UNIX de AT&T [108]. 1, 84

Contramedida: *Software* o *hardware* implementado para mitigar el riesgo potencial. 1

Data URL: Es un esquema que proporciona un método para insertar datos en línea en un documento. Resultan útiles cuando se quiere insertar imágenes o archivos en un documento [70]. 1, 41

DBMS: Abreviación de *Database Manager System*. Es un sistema gestor de base de datos. 1

DLT: Abreviación de *Distributed Ledger Technology* o Tecnología de Contabilidad Distribuida. 1

Entropía: Es una medida de incertidumbre para saber la aleatoriedad en el cifrado de información. 1

Explorador de bloques: Es una herramienta para consultar eventos y operaciones en una blockchain. 1

Exposición: Suma de las vulnerabilidades y su probabilidad de materializarse. 1

Fórmula: Una expresión matemática. 1

Hook: Son funciondes de javascript que permite tanto crear como acceder al estado y al ciclo de vida de React [89]. 1, 58

HTML: Abreviación de *HyperText Markup Language*, lenguaje de marcado para la elaboración de páginas web. 1

HTTP: Abreviación de *Hypertext Transfer Protocol*. Protocolo que permite las transferencias de información a través de archivos en la World Wide Web. 1

HTTPS: *Hypertext Transfer Protocol Secure* (HTTPS) es un protocolo que protege la comunicación y la transferencia de datos entre el navegador web de un usuario y un sitio web [8]. 1, 85

IA: Acrónimo para Inteligencia Artificial. Se aplica cuando una máquina imita las funciones cognitivas que los humanos asocian con otras mentes humanas, como percibir, razonar, aprender y resolver problemas [96]. 1, 5

IP: Abreviación de Internet *Protocol*, usualmente se refiere a una dirección que identifica a un dispositivo conectado a una red. 1

JavaScript: Es un lenguaje de programación interpretado. 1

JSON: JavaScript *Object Notation* es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos JavaScript. [75]. 1, 60

Latex: Es un lenguaje de marcado adecuado especialmente para la creación de documentos científicos. 1

Metamask: Es una billetera de criptomonedas que permite a los usuarios acceder al ecosistema *Web3* de aplicaciones descentralizadas [43]. 1, 58

Modelo: Es la representación de una entidad de nivel de negocio a nivel de código. 1

Neuronas: Tipo de célula que representa la unidad estructural y funcional del sistema nervioso. Su función consiste en transmitir información a través de impulsos nerviosos, desde un lugar del cuerpo hacia otro. [4] En el contexto de ciencias de la computación, hace referencia a un elemento en una red neuronal que toma varios datos de entrada y produce uno de salida. 1

Nodo: Un nodo es un ordenador en la red que aloja y sincroniza una copia de toda la *blockchain*. 1

NoSQL: Es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico SQL. 1

Oráculo: Los oráculos de *blockchain* son entidades que conectan *blockchains* a sistemas externos. 1

POST: El método HTTP POST envía datos al servidor. El tipo de cuerpo de la solicitud se indica mediante el encabezado *Content-Type*. [73]. 1, 86

Protocolo: Es un conjunto formal de estándares y normas que definen la comunicación en una red. 1

Redundancia: Repetición de datos o *hardware* importante que se quiere proteger ante posibles fallos que puedan presentarse. 1

REST: Se trata de un conjunto de restricciones arquitectónicas, no un protocolo o un estándar. Los desarrolladores de API pueden implementar REST de varias formas [90] . 1, 84

RESTful: Transferencia de estado representacional. Es un estilo de arquitectura de software. 1

Riesgo: Probabilidad que tiene una amenaza de ocurrir por medio de una vulnerabilidad. 1

SQL: SQL es un lenguaje de dominio específico, diseñado para administrar información de sistemas de gestión de bases de datos relacionales. 1

Texto cifrado: Texto modificado ilegible que no puede entender ni un humano o computadora. 1

Texto plano: Texto legible que puede ser entendido por cualquier humano o computadora. 1

Tokenizar: Es la acción de convertir un objeto o información confidencial en un token no confidencial y sin valor explotable. 1

URL: Abreviación de *Uniform Resource Locator* o Localizador de Recursos Uniforme. Es una dirección que representa la ubicación de un registro único en la web. 1

V.ºB.º: Abreviatura que quiere decir "visto bueno". 1

Vulnerabilidad: Debilidad de hardware, software o humana que permite tener acceso no autorizado a los recursos. 1

WSGI: WSGI es la interfaz de puerta de enlace del servidor web (*Web Server Gateway Interface* en inglés). Es una especificación que describe cómo un servidor web se comunica con las aplicaciones web y cómo las aplicaciones web se pueden encadenar para procesar una solicitud [110]. 1, 84

XML: Abreviación de *eXtensible Markup Language* o Lenguaje de Marcado Extensible. 1